Bölüm 01

Programlamanın Temelleri

C# Nedir?

Programın Evreleri

Program Yazmaya Giriş

Kaynak Programın Biçemi

Programa Açıklama Ekleme

Girdi/Çıktı İşlemleri

Veri Tipleri ve Değişken Kavramı

Metot Kavramı

Bellek Kullanımı ve Çöp Toplayıcı

Bu kitap hiç programlama bilmeyenler için yazılmıştır.

Bir programlama dilini öğrenmek, anadil öğrenmeye benzer. Çocukken, ana dilimizin sözdizimini (gramer) bilmeden, başkalarının yaptığı gibi kullanmaya başlarız; yani taklit ederiz. Ancak okula başladıktan sonra grameri adım adım öğreniriz. Bu derste genellikle bu yöntemi izleyeceğiz; yani C# dilini sistematik değil, pedagojik öğreneceğiz. Bir çok kavramı önce kullanacak, sonra onların esasını öğreneceğiz.

İlk bölümlerde söylenen kavramları esastan anlamanız beklenmiyor. Programların sözdizimlerinin (syntax - gramer) dayandığı temelleri iyi anlamasanız bile, bir çocuğun ana dilini öğrenmesi gibi, o kalıpları olduğu gibi kullanmaya çalışmalısınız. Yeni başlayanların bir dili öğrenmesinin en iyi yolu budur. Taklit etmek ve bolca tekrarlamak. Bu ders boyunca kavramları sık sık tekrarlayacağız ama her tekrarda o kavrama yeni bilgiler eklemiş olacağız.

Birinci Bölümde Nesne Yönelimli Programlama kavramına doğrudan girmek yerine, her programlama dilinde var olan ve programlamanın temelleri sayılan kavramlar, hiç programlama bilmeyenler için açıklanacaktır. Daha önce bir programlama dili öğrenmiş olanlar, bu bölümü okumadan atlayabilirler.

C# Nedir?

C# dilinin yaratıcısı olan Microsoft onu şöyle tanıtıyor:

C# basit, modern, nesne yönelimli, tip-korumalı ve C ile C++ dillerinden türetimiş bir dildir.

Bazılarına göre C# simgesinin sağındaki # simgesi, C++ simgesinin sağındaki ++ simgelerinin üst üste konmasıyla oluşturulmuştur.

Tabii, bu noktada Microsoft'un söylemediği bir gerçeği biz söylemeliyiz. Yukarıdaki sözlerde C# yerine çekinmeden *java* sözcüğünü koyabilirsiniz. Üstelik, java C# dan çok önce doğduğu için, o sıfatları fazlasıyla hakeder. Ancak, daha sonra yaratılan bir dilin, öncekilerin iyi yanlarını tamamen almış, iyi olmayan yanlarını iyileştirmiş olacağını tahmin edebiliriz. Bu tahmin bile , C# dilinin yeteneklerini anlatmak için yeterlidir.

C# simgesi İngilizce'de "Si-Şarp" diye okunur. C# kolay öğrenilir. C, C++ ve java dillerinin iyi özeliklerini almıştır. Kullanıcı dostu ve hızlı bir uygulama geliştirme (RAD :Rapid Application Development) aracı olduğu kabul edilir.

C# dilinin başlıca avantajları şunlardır:

- Etkileşimli geliştirme aracıdır.
- Windows ve Web uygulamaları için görsel tasarım sağlar.
- Derlenen bir programdır (Scripting dili değildir).
- Debug yapar.

C# dilinin başlıca nitelikleri:

- Basitlik
- Tip-korumalı
- Önceki Sürümleri Destekleme
- Nesne Yönelimli dillerin bütün niteliklerini taşır
- Otomatik çöp toplayıcısı vardır
- Oldukça esnek bir dildir

C# dilinin Başlıca Uygulamaları:

Class Library:- Başka uygulamalarda kullanılacak kütüphane sınıfları yaratır.

Console Application:- Satır komutu arayüzü için görsel C# uygulamaları yaratır.

Asp.Net Web Application:- Web kullanıcı arayüzü için görsel C# uygulamaları yaratır.

Asp.Net Web Service:- XML Web servisleri yaratır.

Asp.Net Mobile Web Application:- PDA, cep telefonları gibi taşınabilir cihazlar için uygulama programları yaratır.

Programlamanın Evreleri

Kullanılan dil ne olursa olsun, programlama eylemi şu evrelerden oluşur:

1. Kaynak programı yazma

Herhangi bir programlama dilinde text olarak yazılır. Kaynak program, kullandığı dilin sözdizimine (syntax) uymalıdır.

2. Derleme

Kaynak programı bilgisayarın anlayacağı bir ara dile dönüştürür. Ara dil kullanılan dile bağlıdır. Örneğin, C dilinde `obj` kodları, java dilinde `bytecode`, C# dilinde `IL`, vb adlar alır.

3. Aradil kodları yürütülebilir makina diline dönüştürülür

Programın yürütülebilir (çalışabilir, koşabilir - *executable*) olabilmesi için, kullanılan İşletim Sisteminin anladığı dile (makina dili) dönüşmesi gerekir. Windows İşletim Sisteminde yürütülebilir programların dosya adları `.exe` uzantısını alır.

Bu bağlamda bir konuyu daha açıklamakta yarar vardır. Çağdaş programcılıkta, her şey bir program içine yapılmaz. Daha önce yaratılmış, kullanılmaya hazır çok sayıda program öğeleri vardır. Bunlar, kullanılan dilin kütüphanesi gibidir. Programcı, kütüphaneden istediklerini alıp, kendi programına yerleştirebilir. Birleştirme yöntemi farklı olsa da her dilde bu olanak vardır. C# dilinin çok geniş bir kütüphanesi vardır. Onu kullanmayı gelecek bölümlerde öğreneceğiz.

Windows İşletim Sisteminde `.exe` uzantılı dosyaları kullanıcı yürütelebilir (koşturabilir).

Örneğin,

```
Deneme.cs
```

adlı bir C# kaynak programı bütün aşamalardan geçip makina diline dönüşünce

```
Deneme.exe
```

adını alır. Bu programı koşturmak için

```
Deneme
```

yazıp Enter tuşuna basmak yetecektir. Aynı işi, Windows'un görsel arayüzünde yapmak için, Windows Explorer ile Deneme.exe dosyasını bulup üstüne tıklamak yetecektir.

Program Yazmaya Giriş

Programlamaya yeni başlayanlar için, öğrenmenin en iyi yolu kitaplarda yazılan küçük programları usanmadan yazmak, derlemek ve çalıştırmaktır. Bu küçücük programlar, bilgisayarın yapabileceği bütün işleri size öğretecektir. Unutmayınız ki, büyük programlar bu küçücük programların birleşmesiyle oluşur. O nedenle, bu derste yazılacak küçük programların, programlamayı öğrenmenin biricik yolu olduğunu aklınızdan çıkarmayınız. Burada izlenecek aşamaların herhangi birisinde yapılacak yanlışlar, öğrenciyi bıktırmamalıdır. Aksine, programlamayı çoğunlukla yaptığımız yanlışlardan öğreniriz. O nedenle, yanlış yapmaktan korkmayınız. Üstelik yeni bir şey öğrenirken bilerek yanlışlar yapıp onun doğurduğu sonuçları görmek eğlenceli ve öğretici olabilir.

Bilgisayarınızda *Visual Studio 2008* kurulu olduğunu varsayıyoruz. Bunu kurmak istemeyenler C# derleyicisi ile de yetinebilirler. Her durumda C# derleyicisini kurulduğunu ve gerekli konfigürasyonların

yapıldığını varsayıyoruz. Eğer bu işler yapılmamışsa, önce kurulum işlemlerini yapmalısınız. Kurulumun nasıl yapılacağı ilgili web sayfalarından görülebilir.

Bu Bölümden sonra *Visual Studio 2008* bütünleşik uygulama geliştirme aracını kullanmaya başlayacağız. Bu aracın uygulama geliştirmede ne kadar yararlı ve kolay kullanılır olduğunu o zaman göreceksiniz. Ama bu aracı hemen kullanmaya başlamak, binanın tepesine asansörle çıkmak gibidir. Aşağı katlarda ne olup bittiğini anlamak için, en aşağıdan başlayıp medivenleri adım adım çıkmalıyız. Başlangıçta atacağımız bu zahmetli adımlar, bizim bilgisayarla nasıl iletişim kurduğumuzu anlamamızı sağlayacaktır.

O nedenle, ilk derslerde hiçbir görsel arayüz kullanmadan, işlerimizi doğrudan doğruya işletim sisteminin komutlarıyla göreceğiz. Böyle yapabilmek için, Windows'un Komut İstemi programını açacak ve Dos (Disk Operating System- Windows İşletim Sisteminin Anahtarları-) komutları yazacağız. Komut İstemi'ni aşağıdaki iki yoldan birisiyle açabilirsiniz:

Başlat -> Çalıştır sekmelerine basınca açılan pencerede Aç: ----- kutucuğuna cmd yazıp entere basınız.

Başlat -> Tüm Programlar -> Donatılar sekmelerinden sonra açılan alt pencereden Komut İstemi 'ne tıklayınız.

Bunlardan birisini yaptığınız zaman, ekranda siyah zemin üzerine beyaz yazılar yazılan Komut İstemi penceresinin açıldığını göreceksiniz. Bu pencereye istediğiniz dos komutunu yazabilir ya da yürütülebilir (executable) herhangi bir programı çalıştırabilirsiniz.

Şimdi yazdığımız programları içine kaydedeceğimiz bir dizin yaratalım. İstediğiniz sürücüde istediğiniz adla bir dizin ya da alt dizin yaratıp C# programlarınızı oraya kaydedebilirsiniz. Programlarımızı C: sürücüsünde yaratacağımız csprg dizini içine kaydetmek isteyelim. Bunun için şu komutları yazacağız:

```
md C:\csprg
cd C:\csprg
```

Bunlardan birincisi istediğimiz dizini yaratır, ikincisi ise o dizini etkin kılar. Basit bir deyişle, sistemin yazma/okuma kafası o dizin içine girer ve orada yazabilir ve okuyabilir. Dizine girdiğimizi, Komut İstemi ekranında beliren

```
C:\csprg>
```

görüngesinden anlarız. Bu simgeler daima Komut İstemi penceresinde görünecektir; etkin olan dizini gösterir. Dos komutlarını c:\csprg> etkin (prompt) simgesinden hemen sonra yazacağız. Şimdi program yazmamıza yardım edecek basit bir editör açalım. Komut İstemi penceresinde edit yazıp Enter tuşuna basınız:

```
edit
```

Eğer editör penceresi açılmazsa, sisteminiz edit.com dosyasını görecek biçimde ayarlı değildir. Bunu Ek_A da anlatıldığı gibi ayarlayabilirsiniz. Ama, şimdi zaman kaybetmemek için, edit.com dosyasının bulunduğu adresi tam yazarak, editörü açabiliriz:

```
C:\WINDOWS\system32\edit
```

Açılan editörde aşağıdaki programı yazıp Dosya sekmesinden Farklı Kaydet seçeneğine basınız. Açılan alt pencerede Dosya Adı yerine Program001.cs yazıp Enter tuşuna basınız. İsterseniz yazma ve kaydetme işini Notepad ile de yapabilirsiniz.

Program001.cs

```
class Program001
{
}
```

İpucu

C# kaynak programlarının dosya adlarının uzantısı hiç olmayabileceği gibi, istenen bir uzantı da yazılabilir. Ama, bir dosyanın C# kaynak dosyası olduğunu ilk bakışta anlayabilmemiz için .cs uzantısını yazmayı alışkanlık edinelim.

Program adı ile class adı aynı olmak zorunda değildir. Ama bu derste yazılan örneklerde, hangi sınıfın hangi programda olduğunu kolayca bulabilmek için, program adı ile sınıf adını aynı yapacağız. Bu yöntem *java* dilinde zorunludur.

C# nesne yönelimli bir dil olduğu için, hemen her şey sınıflar (nesneler) içine yazılır. Programda kullanacağımız değişkenler ve metotlar mutlaka bir ya da birden çok sınıf (nesne) içinde tanımlanmalıdır. Bunların nasıl olacağı, bu dersin asıl konusudur ve onları adım adım öğreneceğiz.

Yukarıda yazdığımız Program001.cs dosyası içinde tanımlanan Program001 sınıfının içi boştur. Dolayısıyla, bu sınıfın hiçbir işe yaramayacağı açıktır. Böyle olduğunu, C# derleyicimiz bize söyleyecektir. Bu programı derlemek için

```
csc Program001.cs
```

yazıp Enter tuşuna basınız. Eğer sisteminiz csc.exe ile adlandırılan C# derleyicisini görmüyorsa, derleyicinin bulunduğu adresi yazabilirsiniz.

```
C:\WINDOWS\Microsoft.NET\Framework\v3.5\csc Program001.cs
```

Derleyici size aşağıdaki iletiyi gönderecektir. (İletideki sürüm numaraları, sizin sistemde farklı olabilir.)

```
Microsoft (R) Visual C# .NET Compiler version 7.10.6001.4
for Microsoft (R) .NET Framework version 1.1.4322
Copyright (C) Microsoft Corporation 2001-2002. All rights reserved.


error CS5001: Program 'c:\csprg\Program001.exe' does not have an entry point defined
```

Bu iletinin ilk üç satırı, derleyiciye özgü olan ve telif haklarıyla ilgili bilgilerdir. Bizim için önem taşıyan ileti, error ile başlayan son satırdır. Bunu derin incelemeye gerek yoktur. Bize, Program001.cs dosyasını derleyip Program001.exe adlı yürütülebilir dosyayı yaratamadığını söylüyor. Bunu yapamayış nedenini, bir giriş noktası olmamasına bağlıyor. Bu hatayı da CS5001 numaralı hata olarak işaretliyor.

İpucu

Derleyici kaynak programda sözdizimi (syntax) hatası olduğunda programı derleyemez. Derleyici, derleyemediği her programda, sözdizimi yanlışlarını türlere ayırır ve her yanlış türüne bir kod numarası verir. Bu kod numaraları, programın düzeltilmesinde (debug) çok işe yararlar.

Ama, şimdilik bunları hiç düşünmeden, derleyicinin bizden istediği giriş noktasını oluşturmaya çalışalım.

C# programları bir `Main()` fonksiyonu tarafından eyleme geçirilir. Programın yapacağı bütün işler bu Main() fonksiyonu tarafından belirlenir. Buna programın giriş noktası diyoruz. Şimdi yukarıdaki programımıza `Main()` fonksiyonunu ekleyelim ve Program002.cs adıyla kaydedelim.

Program002.cs

```
class Program002
{
      static void Main()
      {
      }
}
```

Sonra programı derleyelim:

```
csc Program002.cs
```

Oley! Bu kez derleyicimiz hiç hata (error) iletisi göndermedi. Demek ki programı derledi ve yürütülebilir makina diline çevirdi. Gerçekten

```
dir
```

komutu verirseniz, c:\csprg dizini içinde Program002.exe adlı yürütülebilir bir program olduğunu göreceksiniz. Bu dosyayı koşturmak için

```
Program002
```

komutunu yazmanız yetecektir. Bunu yazdığınızda, program çalışır, ama siz ekranda bir şey göremezsiniz. Çünkü, programın içine ekrana çıkacak bir şey yazmadık. Şimdi bunu yapalım.

İpucu

Bir dili öğrenirken işin pedagojisi ile sistematiği asla paralel gitmez. Bu nedenle, bazen pedagojiden bazen sistematikten sapmak gerekir. Böyle durumlarda, görünmez uzaylı dostumuz Uzay bize ipucu verecektir. Şimdilik Uzay'ın bize verdiği ipuçlarını kullanalım. Zamanla, onları daha iyi kavrayacağız.

İpucu

Ekrana bir şey yazdırmak için `Write()` ya da `WriteLine()` fonksiyonları kullanılır. İkisi de aynı işi yapar, ancak birincisi isteneni yazdıktan sonra, yazdığı şeyin sonunda bekler, ikincisi isteneni yazdıktan sonra alttaki satırın başına geçer.

Şimdi bu ipucuna göre programımızı düzeltelim.

Program003.cs

```
class Program003
{
    static void Main()
    {
        WriteLine("Merhaba C# ")
    }
}
```

Bu programı kaydettikten sonra derleyelim:

```
csc Program003.cs
```

Derleyicimizin, bu kez,

```
Program003.cs(5,27): error CS1002: ; expected
```

hata iletisini verdiğini göreceğiz. Bu ileti kaynak programın 5-inci satırının 27-inci kolonunda (;) beklendiğini söylemektedir.

İpucu

C# dilinde her deyimin sonuna (;) konulur. CS1002 hata kodu, kaynak programda bir deyimin sonuna (;) konmadığında oluşur.

Derleyicinin hata iletisi kesin olmakla birlikte, ekran çözünürlüğüne bağlı olarak, hatanın oluştuğunu işaret ettiği yer (kolon) bize farklı görünebilir. O zaman, hatayı işaret edilen yere yakın yerlerde aramalıyız.

Şimdi eksik olan (;) yazarak programımızı düzeltelim (bu işleme 'debug' denir).

Program004.cs

```
class Program004
{
    static void Main()
    {
        WriteLine("Merhaba C# ");
    }
}
```

Bu programı Program004.cs adıyla kaydedelim ve derleyelim:

```
csc Program004.cs
```

Derleyicimiz başka bir hata iletisi gönderir:

```
Program004.cs(5,3): error CS0103: The name 'WriteLine' does not exist in the class or namespace 'Program004'
```

Derleyicimiz bu kez, 5-inci satırın 3-üncü kolonunda hata bulmuştur. 'WriteLine' nın ne olduğunu Program004 içinde bulamamıştır.

İpucu

C# dilinde her değişken ve her metot (fonksiyon) mutlaka bir sınıf içinde tanımlanır. `Write()` ve `WriteLine()` metotları `Console` sınıfı içinde tanımlıdırlar.

Öyleyse, derleyicimize `WriteLine()` metodunu `Console` sınıfı içinde bulacağını söylemeliyiz. Bunu metodun önüne sınıfın adını koyarak yapıyoruz. Sınıf ile metot arasına (.) konulur. `Console.Writeline()` yazdığımızda, derleyici, `WriteLine()` metodunun `Console` sınıfı içinde olduğunu anlayacaktır.

Derleyicimizin uyarıları doğrultusunda kaynak programdaki eksiklerimizi tamamlıyor ve yanlışlarımızı düzeltiyoruz. Bu yaptığımız iş gerçek anlamda bir 'debug' işlemidir. Programcılıkta çok önemli olan bu işi erinmeden yapmalıyız. Yaptığımız yanlışlar bize çok şey öğretecektir.

Program005.cs

```
class Program005
{
    static void Main()
    {
        Console.WriteLine("Merhaba C# ");
    }
}
```

Bu programı Program005.cs adıyla kaydedelim ve derleyelim. Artık, derleyicinin her istediğini yaptığımıza göre, kaynak programımızın derlenebilmesini umut etmekteyiz.

```
csc Program004.cs
```

Heyhat! Derleyicimiz gene hata iletisi gönderiyor:

```
Program005.cs(5,3): error CS0246: The type or namespace name 'Console' could not be found (are you missing a using directive or an assembly reference?)
```

hata gene 5-inci satırda oluşuyor. Derleyicimiz 'Console' un tanımını bulamıyor ve ipucu veriyor. Console'un bulunduğu yeri bir using directive ya da bir assembly reference olarak vermeyi unutmuş olabileceğimiz olasığı nı belirtiyor. Bu sorunu çözmek için Uzay'dan ipucu istemeliyiz.

Uzay dostumuz bize `WriteLine()` metodunun `Console` sınıfı içinde olduğunu, `Console` sınıfının da System namespace'i içinde olduğu ipucunu verdi. O halde, derleyicimizin istediği bilgileri ona verebiliriz.

Program006.cs

```
class Program006
{
    static void Main()
    {
        System.Console.WriteLine("Merhaba C# ");
    }
}
```

Artık son düzeltmeyi yaptığımızı umarak bu programı Program006.cs adıyla kaydedip derliyoruz:

```
csc Program006.cs
```

Oleeey! Derleyicimiz başka hata vermedi, programımız derlendi. Gerçekten

```
dir
```

komutunu yazarsak, C:\csprg dizininde Program006.exe yürütülebilir dosyasının yaratıldığını görebiliriz. Şimdi bu dosyayı koşturmak için,

```
Program006
```

yazıp Enter tuşuna basarsak, ekranda

```
Merhaba C#
```

yazısını göreceğiz.

Buraya kadar altı adımda ekrana 'Merhaba C#' tümcesini yazdıran bir program yazdık. Bu bir tümce yerine bir roman ya da karmaşık matematiksel işlemlerden oluşan deyimler de olabilirdi. Uzun ya da kısa metinler veya basit ya da zor işlemler için yapacağımız iş hep budur. Bu biçimde yazacağımız küçük programlarla C# dilinin bütün hünerlerini öğreneceğiz. Büyük programların bu küçük programların uyumlu birleşmesinden oluştuğunu hiç aklımızdan çıkarmayalım. O nedenle, Visual Studio 2008 uygulama geliştirme aracını kullanmaya başlamadan, C# dilinin temellerini öğrenmeye devam edeceğiz.

Kaynak Programın Biçemi

Programlama dillerinde, belirli bir işi yapmak üzere yazılan sözdizimi (kod) parçasına deyim denir. Örneğin, System.Console.WriteLine("Merhaba C# "); bir deyimdir. C# dilinde bir deyimin bittiğini derleyiciye bildirmek için, o deyimin sonuna (;) konulur. Bir deyimdeki farklı sözcükler birbirlerinden bir boşluk karakteri ile ayrılır. Kaynak programda ardışık yazılan birden çok boşluk karakterleri tek boşluk olarak algılanır. Tablar, satırbaşları ve boş satırlar birer boş karakter olarak yorumlanır. Dolayısıyla, yukarıdaki programı

Program006a.cs

```
class Program006a { static void Main(){ System.Console.WriteLine("Merhaba C# ");}}
```

biçiminde ya da

Program006b.cs

```
class
Program006b
{
static


void
Main()


{
System.Console.WriteLine("Merhaba C# ");
}
}
```

biçiminde de yazabiliriz. Derleyici, kaynak programı derlerken birden çok tekrar eden boşluk karakterlerini, tab ve satırbaşlarını yok sayacaktır.

Buraya kadar C# diline ait bazı kavramları açıklamadan kullandık. Bu noktada basit açıklamalar yapmak yararlı olabilir.

Aduzayı (namespace)

C# kütüphanesinde çok sayıda metot ve değişken tanımları vardır. Metotların ve değişkenlerin her biri bir sınıf içindedir. Metotları ve değişkenleri içeren sınıfların sayıları da çok fazladır. Erişimi kolaylaştırmak için sınıflar gruplara ayrılmışlardır. Birbirleriyle ya da aynı konuyla ilişkili olan sınıflar bir grup içinde toplanır. Bu grupların her birine aduzayı (namespace) diye adlandırılan birer ad verilir. Başka bir deyişle, bir aduzayı değişken, fonksiyon, sınıf vb varlıkların adlarını içeren soyut bir ambardır (container).Örneğin `sistem` ile ilgili olan bütün sınıflar `System` aduzayı (namespace) içindedir. Bir aduzayında aynı adı taşıyan iki sınıf olamaz. Ama farklı aduzayları içinde aynı adı taşıyan sınıflar olabilir. Aduzay adları bu sınıfların karışmasını önler. Bir aduzayı (`namespace`) içerdiği varlıkları lojik olarak gruplar; yani grup üyelerinin fiziksel kayıt ortamında aynı yerde olmaları gerekmez.

Sınıflar (class)

Sınıflar (class) Nesne Yönelimli Programlamanın (Object Oriented Programming) temel taşlarıdır. Programın kullanacağı her değişken, her deyim, her metot (fonksiyon) mutlaka bir sınıf içinde tanımlanmalıdır. Bir sınıf içinde tanımlanan metotlar (fonksiyonlar) ve değişkenler o sınıfın `öğe`'leridir (member). Karışmamaları için, bir sınıfta aynı adı taşıyan iki öğe olamaz (aşkın öğeler kavramını ileride göreceğiz).

Uzay dostumuz bize `WriteLine()` metodunun `Console` sınıfı içinde olduğunu, `Console` sınıfının da System namespace'i içinde olduğu ipucunu verdi. O halde, derleyicimizin istediği bilgileri ona verebiliriz.

Bloklar

Kaynak program bir ya da bir çok sınıftan oluşur. Sınıfların her biri bir bloktur. Bir sınıf içindeki her metot (fonksiyon) bir bloktur. Örneğin, yukarıdaki programın tamamından oluşan

```
class Program006
{
}
```

bir bloktur. Benzer olarak,

```
static void Main()
{
}
```

metodu da bir bloktur. `Main(){ }` bloku, class Program006.cs { } blokunun içindedir. Bu tür bloklara iç-içe bloklar diyoruz. Bir blok içinde gerektiği kadar iç-içe bloklar oluşturabiliriz. Bir programda birden çok sınıf ve bir sınıf içinde birden çok blok olabilir. Bloklar, adına blok parantezleri diyeceğimiz { } simgeleri içine yazılır. Bazan Main(){ } blokunda olduğu gibi blok parantezlerinin önüne blok adı gelebilir. Her sınıf (class) ve her alt sınıf bir bloktur. Bir sınıf içinde yer alan her metot (fonksiyon) bir bloktur. İleride göreceğimiz if, case ve döngü yapılarının her birisi bir bloktur. Bir sınıf içinde birbirlerinden bağımsız bloklar seçkisiz sırada alt-alta yazılabilir. Ancak iç-içe olan bloklar, istenen işlem sırasını izleyecek uygun sırayla iç-içe konur. İç-içe blok yazılırken, en içten dışa doğru blokun başladığı ve bittiği yerleri belirten blok parantezlerinin ({ }) birbirlerini karşılaması sağlanmalıdır. Bu yapılmazsa derleme hatası doğar. İç-içe bloklarda, işlem öncelikleri en içten en dışa doğru sıralanır.

Derleyici kaynak programın biçemine (format) değil, sözdizimine bakar. Ama kaynak programı hem kendimiz hem de başka programcılar için kolay okunur biçimde yazmak iyi bir alışkanlıktır. İç-içe giren blokları birer tab içeriye almak, alt-alta yazılan bloklar arasına birer boş satır koymak, kaynak programın, programcı tarafından kolay okunup anlaşılmasını sağlar. Özellikle, uzun programlarda düzeltme (debug) ya da güncelleme (update) yaparken, kaynak programın yazılış biçemi işin kolay ya da zor yapılmasına neden olur.

Programa Açıklama Ekleme

Kaynak programımızın biçemi bilgisayar için değil, onu okuyan kişiler için önemlidir, demiştik. Büyük bir programın kaynağının kolay anlaşılır olmasını sağlamak programcının ahlâki sorumluluğundadır. Bunu sağlamak yalnızca programın biçemiyle olmaz. Büyük programlar için sınıfların, metotların, değişkenlerin işlevleri ayrıca açıklanmalıdır. Büyük yazılım firmaları programdaki her ayrıntıyı açıklayan döküman hazırlarlar. Böylece, yıllar sonra programcılar değişse bile, o dökümanlara bakılarak, kaynak program üzerinde değişiklikler, iyileştirmeler yapılabilir.

C# dilinde programa açıklama eklemek için iki yöntem kullanırız. Bu yöntemler C, C++ ve java gibi bir çok dilde de var olan yöntemlerdir.

Tek satır açıklaması

// simgesinden sonra satır sonuna kadar yazılanlar derleyici tarafından işlenmez. Bir satırın bütününü ya da bir satırın ortasından sonuna kadar olan kısmı açıklamaya koyabiliriz:

```
// Bu metot net ücretten gelir vergisini hesaplar
// Console.WriteLine("Merhaba dünya!");
return netÜcret;    // metodun değeri netÜcret 'tir
```

Birincide satırın tamamı bir açıklamadır. Örneğin, gelir vergisi hesaplayan bir metodun başladığı satırdan önceki satıra konulabilir, böylece metodun ne iş yaptığı her okuyan tarafından anlaşılır.

İkincide, genellikle program yazarken bazı deyimlerin programa konulup konulmamasının etkisini görmek için, programcının sık sık başvurduğu bir yöntemdir. Bir deyimin başına // simgelerini koyarak o deyimi derleyicinin görmesini engelleyebilirsiniz.

Üçüncüde, önce bir deyim yazılmış, deyimin bittiği yerde // simgesi konulmuştur. Derleyici deyimi görür ve işler, ama // simgesinden sonra satır sonuna kadar yazılanları görmez. Bu yöntemle, bir deyimin ne iş yaptığını kolayca açıklayabiliriz.

Çoklu satır açıklaması

Bazan yapacağımız açıklama tek satıra sığmayabilir. O zaman ardışık açıklama satırlarını /* */ simgeleri arasına alırız.

```
/*
Bu sınıf 20 Ağustos 2008 tarihinde
C#  ile Nesne Programlama
kitabının Onikinci Bölümü için yazılmıştır.
*/
```

Bu yöntem çok satırlı açıklamalar için kullanıldığı gibi, program yazarken ardışık satırlardan oluşan bir blokun, bir metodun, bir sınıfın programda olup olmamasının etkisini görmek için programcının geçici olarak başvurduğu önemli bir araçtır.

Girdi-çıktı İşlemleri

Kullanıcının bilgisayarla etkileşim içinde olabilmesi için, onunla iletişim kurabilmesi gerekir. Bunun tek yolu, bilgisayara `girdi (input)` göndermek ve ondan `çıktı (output)` almaktır. Hemen her dil bu

iş için çeşitli yöntemlere sahiptir. `Giriş/çıkış` birimleri dediğimiz birimler bu işe yarar. Örneğin, klavye, dosya, uzaktaki bilgisayar, ağ gibi araçlar bilgisayara girdi yapabilirler. Ekran, yazıcı, dosya, ses-aygıtları, ağ gibi araçlarla bilgisayardan çıktı alınabilir. Her gün sayısı ve niteliği değişen bu giriş-çıkış aygıtlarını kullanmakta C# çok hünerlidir. Onları ilerideki derslerde yeri geldikçe ele alacağız. Ama ilk derslerde çok kullanacağımız bazı kavramları tanımamız gerekiyor.

System aduzayı

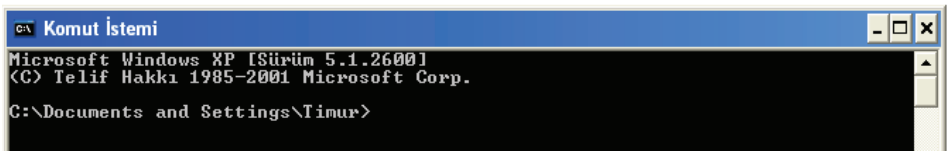
Bilgisayar sistemiyle ilgili sınıfları içeren bir aduzayıdır (`namespace`).

Console

`System` aduzayı içinde bir sınıftır. Konsol (console) uygulamaları için standart `giriş`, `çıkış` ve `hata` akışlarını belirler. `Console` sınıfından kalıtım (inherite) elde edilemez. Bildirimi şöyledir.

```
public static class Console
```

Bunların ne anlama geldiğini şimdilik merak etmenize gerek yoktur. Nesne yönelimli programlamanın alfabesi olan bu kavramlarla dersin ilerleyen evrelerinde karşılaşacak ve ne olduklarını kolayca anlayacağız. `Konsol (console)`, işletim sisteminin kullanıcıya açtığı bir etkileşim penceresidir. Bu pencere aracılığıyla kullanıcı ile sistem arasında iletişim kurulur. Kullanıcı klavyede yazdıklarını konsolda görebilir. Sistem kullanıcının konsola yazdığı komutları algılar ve ona yanıtları (tepki) gene konsola text tipi çıktı göndererek verir. Windows işletim sisteminde konsol, adına MS-DOS komut ekranı (command prompt) denilen şu penceredir.



Bu pencereden DOS komutları girilebilir ve sistem aynı pencereden kullanıcıya text tipi çıktılarla yanıt verir.

Konsol G/Ç Akım-yolları (Console I/O streams)

Bir konsol uygulaması başladığında, işletim sistemi kendiliğinden şu üç giriş/çıkış akım-yolunu konsola bağlar:

- In
- Out
- Error

Kullanacağımız Windows işletim sisteminde standart giriş yolu klavye, standart çıkış yolu ekrandır. İstenirse bunlar değiştirilebilir. Örneğin klavyeden giriş yerine bir dosyadan giriş yapılabilir. Benzer olarak, ekran yerine yazıcıya veya bir dosyaya çıkış yapılabilir. Ama biz bu ders boyunca standart giriş birimi olarak klavyeyi, standart çıkış birimi olarak ekranı kullanacağız. Giriş `In`, çıkış `Out` ve hata `Error` akım-yollarıyla temsil edilir. Bunlar konsol sınıfında `girdi`, `çıktı` ve `hata` 'yı tutan değişkenlerdir. Bu değişkenlere nesne yönelimli programlama dillerinde, değişken demek yerine, `gendeğer` (property) denilir. Bazı kaynaklar "özellik" sözcüğünü kullanır. Bu kitapta `gendeğer` sözcüğünü tercih edeceğiz.

Yukarıdaki üç gendeğerin değerlerinin nasıl oluştuğunun ayrıntısına girmeden, konuyu basitleştirmek için şunu söyleyebiliriz. Bir konsol uygulaması başlayınca, kullanıcının konsola yazdıkları `In` akım-yoluna atanan değerdir. Sistem bu girdiyi algılar. Sistemin çıktıları `Out` akım-yoluna atanan değerdir. O değer konsola yazılır ve kullanıcıya iletilmiş olur. Bu işlemler sırasında bir hata oluşursa, o hata `Error` akım-yoluna atanır ve otomatik olarak konsola yazılır.

Girdi/Çıktı için Metotlar

C# dilinde girdi/çıktı işlemleri için `Console` sınıfına ait olan aşağıdaki metotları (fonksiyon) çok kullanacağız. O nedenle, onlar hakkında basit temel bilgileri edinmemiz gerekiyor.

Nesne Yönelimli Programlama dillerinde, fonksiyon kavramı matematikteki fonksiyon kavramından biraz daha geneldir. O nedenle, `fonksiyon` terimi yerine `metot` terimi kullanılır.

Write()

`Console` sınıfı içinde bir metottur (fonksiyon). Konsola çıktı gönderir. Sözdizimi:

```
Console.Write(parametre);
```

Bu metot parametre olarak C# dilindeki her veri tipini kabul eder.

WriteLine()

`Console` sınıfı içinde bir metottur (fonksiyon). Konsola çıktı gönderir. `Write()` metodu ile aynı işi yapar, ancak çıktıyı gönderdikten sonra ayrıca satırbaşı yapar (\r\n). Sözdizimi:

```
Console.WriteLine(parametre);
```

Read()

`Console` sınıfı içinde bir metottur . Standart giriş akım-yolundan gelen bir sonraki karakteri (henüz buffer'a girmeyen ilk karakteri) okur. Bu demek, eğer kullanıcı klavyeden giriş yapıyorsa, kullanıcının girdiği ilk karakteri okur. Arka arkaya `Read()` metodu kullanılıyorsa ~~yapılıyorsa~~, standart giriş akımından gelen karakterler geliş sırayla okunur. Sözdizimi:

```
Console.Read();
```

ReadLine()

`Console` sınıfı içinde bir metottur. Standart giriş akım-yolundan gelen sonraki satırı okur. Burada sonraki satır derken, henüz buffer'a girmeyen ilk satır kastedilmektedir. Arka arkaya `ReadLine()` metodu kullanılıyorsa, standart giriş akımından gelen satırlar geliş sırayla okunur. Sözdizimi:

```
Console.ReadLine();
```

Uyarı

Read() metodu bir karakter okur, `ReadLine()` metodu bir satır okur. Dolayısıyla, `Read()` metodunun değeri bir karakter, `ReadLine()` metodunun değeri bir `string`'dir. O nedenle, sayısal verileri `ReadLine()` metoduna `string` olarak okutur, sonra gerçek sayı tipine dönüştürürüz. Bu dönüşümü `Parse()` metodu ile yaparız..

Parametre

Bilgisayar programlarında, fonksiyonun bağlı olduğu değişkeni öteki değişkenlerden ayırmak için, onlara parametre denilir. Matematikte f(x) fonksiyonu için x değişkeni ne anlam taşıyorsa, bilgisayar fonksiyonları için parametre aynı anlamı taşır.

Parse()

Kullanıcının girdiği sayısal veri `ReadLine()` metodu tarafından string olarak okunur. Okunan bu veriyi olması gereken sayısal veri tipine dönüştürür. Sözdizimi:

```
string s;
```

```
int n;
```

değişken bildirimi yapılmış iken

```
s = Console.ReadLine(s);  n = int.Parse(s);
```

ya da `s` değişkenini kullanmadan

```
n = int.Parse( Console.ReadLine() );
```

yazılabilir.

.NET Framework C# veri tiplerini tanır ve onu kendi veri tipine dönüştürür. Dolayısıyla, C# veri tipleri yerine .NET teki karşılığını kullanabiliriz. Bu dönüşümü Veri Tipleri ve Değişkenler bölümde göreceğiz. Şimdilik, `int.Parse()` yerine `Int32.Parse()` yazabileceğimizi söylemekle yetinelim.

Veri Tipleri, Değişkenler ve Metotlar

Programlama dillerinde değişkenlerin ve metotların (fonksiyon, prosedür) işlevleri çok büyüktür. Değişkenler programın ham maddeleridir. Metotlar ise bu ham maddeleri işleyen araçlardır. Bunlarla ilgili ayrıntıları ilerideki derslere bırakıp, burada veri tipi, değişken ve metot kullanımına basit örnekler vereceğiz.

Program007.cs

```
class Program007
{
    static void Main()
    {
        int puan;
        puan = 19;
        System.Console.WriteLine(puan);
    }
}
```

Bu programı Program007.cs adıyla kaydedip derleyiniz ve koşturunuz. Çıktının

```
19
```

olduğunu göreceksiniz.

Bir programda farklı veri tipleriyle işlem yapmamız gerekebilir. Örneğin, tamsayılar, kesirli sayılar, karakterler (harfler ve klavyedeki diğer simgeler), metinler (string), mantıksal (boolean) değerler (doğru=true, yanlış=false) ilk aklımıza gelen farklı veri tipleridir. Bu farklı veri tiplerinin büyüklükleri (bellekte kaplayacakları yer) ve onlarla yapılabilecek işlemler birbirlerinden farklıdır. Örnekse, sayılarla dört işlem yapabiliriz, ama metinlerle yapamayız. O nedenle, C# ve başka bazı diller verileri tiplere ayırır. Değişken tanımlarken onun hangi tip veriyi tutacağını belirtir. Böylece, ana bellekte o değişkene yetecek bir yer ayırır ve o veri tipine uygun işlemlerin yapılmasına izin verir. Kabaca söylersek, değişken, ana bellekte belirli tipten bir veriyi tutması için ayrılan adrestir.

Aslında, C# dilinde her veri tipi bir sınıftır. Tersine olarak, her sınıf soyut bir veri tipidir. Bu konuyu ileride daha ayrıntılı inceleyeceğiz. Şimdilik, *int, char, string, bool vb.* temel veri tiplerini açıklamasız kullanacağız.

Şimdi bu programı satır satır inceleyelim. Bu basit işi yapmakla, bundan sonra yazacağımız programlardaki benzer ifadelerin ne iş yaptığını öğrenmiş olacağız.

1. Satır: Bu satır Program007 adlı bir sınıf tanımlamaya başlar.

2. Satır: { simgesi sınıf blokunun (sınıf gövdesi) başlangıcıdır.

3. Satır: `void Main()` ifadesi `Main()` metodunun (fonksiyon) belirtkesidir. Bu metodun adının Main, değerinin void (boş) olduğunu ve hiçbir parametreye (değişken) bağlı olmadığını belirtir. Bu satırın başındaki static deyiminin ne işe yaradığını dersin ilerleyen bölümlerinde öğreneceksiniz. Şimdilik size bir şey ifade etmese bile, şu basit açıklamayı yapacağız. Bir sınıfta `static` nitelemeli değişkenler ve metotlar, o sınıfa ait bir nesne yaratmadan kullanılabilirler.

4. Satır: { simgesi `Main()` metodunun blokunun (gövde) başlama yeridir.

5. Satır: Bu satırında yer alan int = puan; ifadesi bir deyimdir. Bu deyim iki iş yapar:

puan adıyla bir değişken tanımlar,

Bu değişkenin tutacağı verinin int (tamsayı) tipinden olacağını derleyiciye bildirir. Derleyici ana bellekte puan değişkenine verilecek tamsayı değerlerin sığabileceği büyüklükte bir yer ayırır. Bu yere puan değişkeninin bellekteki adresi denir. 'puan' adı o adresi işaret eden bir referanstır. Bazı dillerde buna pointer denilir. Ancak, C# , kullanıcıya pointer kullanma izni vermediği için, 'pointer' yerine 'referans' sözcüğü kullanılır.

6. Satır: Bu satırda yer alan puan = 19; ifadesi bir atama deyimidir. Önceki deyimde tanımlanan puan değişkenine 19 tamsayı değerini atar. Başka bir deyişle, puan değişkeninin bellekteki adresine 19 yazılır.

7. Satır: Bu satırda yer alan System.Console.WriteLine(puan); ifadesi puan değişkeninin değerini ekrana yazar.

8. Satır: } simgesi `Main()` metodunun bitiş yeridir.

9. Satır: } simgesi Program007 sınıfının bitiş yeridir.

Başlamışken, program üzerinde küçük değişiklikler yaparak, o değişikliklerin etkilerini görmeye çalışalım.

Atama Deyimi

Bir değişkene değer atamak demek, o değişkene ana bellekte ayrılan adrese bir değer girilmesi demektir.

```
int puan;
```

değişken bildirimi yapılmışken,

```
puan = 19;
```

deyimi bir atama eylemidir. Puan adlı değişkene 19 değerini verir. Atanan değer, değişken için ana bellekte ayrılan adrese yerleşir.

Yukarıdaki iki deyimi birleştirip tek deyim haline getiriniz.

```
int  puan = 19;
```

Programı bu biçimiyle derleyip koşturduğunuzda, aynı çıktıyı verdiğini göreceksiniz. Demek ki, değişken tanımını ve değer atamayı ayrı ayrı deyimlerle yapabileceğimiz gibi, değişken tanımını ve ilk değerini atamayı tek bir deyim ile de yapabiliriz. Ohalde,

```
int puan;
puan = 19;
```

deyimleri yerine

```
int  puan = 19;
```

deyimini yazabiliriz.

Şimdi programın yedinci satırını şöyle değiştirelim:

Program007a.cs

```
class Program007a
{
     static void Main()
     {
           int puan;
           puan = 19;
           System.Console.WriteLine(puan + 2);
     }
}
```

Bu programın çıktısı

```
21
```

dir. Demek ki WriteLine() metodumuz işlem yapabilmektedir.

Alıştırma

Bu metodun başka bazı hünerlerini görmek için, yedinci satır yerine, sırasıyla, aşağıdakileri yazarak programı her birisi için ayrı ayrı koşturunuz.

```
System.Console.WriteLine("Ankara Türkiye'nin başkentidir.");
System.Console.WriteLine(puan - 2);
System.Console.WriteLine(puan * 2);
System.Console.WriteLine(puan / 2);
System.Console.WriteLine(puan % 2);
System.Console.WriteLine(123 + 456);
System.Console.WriteLine(12*2 + 2 - 8/4);
```

Parametre ve Değişken

Matematik derslerinden anımsayınız. Bir fonksiyonu tanımlarken, onun değişkenlerini `f(x,y,z)` biçiminde yazarız. Bilgisayar programlarında da benzer işi yaparız. Fonksiyon adından sonra ( ) içine fonksiyonun bağlı olduğu değişkenleri yazarız. Bu değişkenleri, sınıf içinde bildirimi yapılan öteki değişkenlerden ayırmak için, 'değişken' yerine 'parametre' sözcüğünü kullanırız, demiştik. Buna ek olarak, yukarıdaki örneklerden görüyoruz ki, parametre bir işlem de olabiliyor. Bu durumuyla, parametre kavramı matematikteki değişken kavramına göre daha işlevseldir. Bu işlevsellik ilerideki konularımızda daha da artacaktır. Örneğin, yukarıdaki alıştırmalarda kullanılan WriteLine() metotlarının parametreleri, sırasıyla, şunlardır:

```
Ankara Türkiye'nin başkentidir.
puan -2
puan * 2
puan / 2
puan % 2
123 + 456
12*2 + 2 – 8/4
```

Bunların hepsinde parametre bir tanedir. Çıktılar, parametrelerin bellekteki değerleridir.

Metot Kavramı

Önemi nedeniyle, metot kavramını biraz daha açmamız gerekiyor. Gerektiğinde bir metotta birden çok parametre kullanabiliriz. Örneğin, aşağıdaki program `Hesap` adlı bir sınıf, sınıfın içinde `sonuç` adlı `int` tipinden bir değişken ile `Hesapla()` ve `Main()` metotlarını tanımlıyor.

Hesap.cs

```
using System;
class Hesap
{
    static int sonuç;
    static int Hesapla(int a, int b)
    {
        int x, y;
        x = 3 * a;
        y = b / 2;

        return x + y;
    }

    public static void Main()
    {
        int n;
        n = Hesapla(4, 8);
        sonuç = n;
        Console.WriteLine(sonuç);
    }
}
```

Bir programda yer alabilecek örnek öğeleri içerdiği için, programı satır satır çözümleyerek konuyu daha iyi anlatabiliriz. Burada yapacağımız basit açıklamaların gerisinde, C# dilinin sistematik yapısı yatmaktadır. O sistematik yapıyı ilerleyen derslerde adım adım açıklayacağız. Şimdi söyleyeceğimiz kavramları esastan anlamanız beklenmiyor. Programların sözdizimlerini (syntax -gramer) algılamaya çalışmanız ve onları şablon gibi kullanmanız yetecektir.

Baştan sona doğru açıklamaya başlayalım.

`using System;` deyimi `System` aduzayını (namespace) çağırıyor. Hemen her uygulamada bunu yapmamız gerekiyor.

`class Hesap` satırı Hesap adlı bir sınıf (class) bildirimini başlatıyor. Sınıfın bütün öğeleri en dıştaki {} bloku içinde yer alır. Bu bloka sınıf bloku veya sınıf gövdesi denilir. C# dilinde global değişken ve global metot yoktur. Bütün değişkenler ve metotlar sınıflar içinde tanımlanır. Bir sınıfın gövdesinde yer alan değişkenler ve metotlar o sınıfın öğeleridir (class member). Sınıfın öğelerine sınıf içinden doğrudan erişilebilir. Sınıfın öğelerine sınıf dışından erişmek için sınıf adı veya sınıfın bir nesnesi referans alınır.

`Static int sonuç;` deyimi sınıfa ait `sonuç` adlı int tipinden bir değişken tanımlar. Bu tür değişkenlere *sınıf değişkeni* ya da *sınıf öğesi* diyeceğiz. `sonuç` değişkeni `static` nitelemelidir. `static` nitelemeli değişkenleri ve metotları, sınıfa ait bir nesne yaratmadan kullanabileceğimizi söylemiştik.

```
    static int Hesapla(int a, int b)
    {
        int x, y;
        x = 3 * a;
        y = b / 2;

        return x + y;
```

```
    }
```

bloku `Hesapla()` adlı bir metot (fonksiyon) bildirimidir. Bir sınıf içinde bildirimi yapılan metot o sınıfın bir öğesidir. Metot adının sonuna gelen () parantezleri derleyiciye, yapılan bildirimin bir metot olduğunu bildirir. Bu parantezin içindeki `(int a, int b)` ifadeleri, metodun `int` tipinden `a` ve `b` adlı iki parametreye (değişken) bağlı olduğunu belirtir. Metot kullanılmak üzere çağrılırken `a` ve `b` parametrelerine istenen gerçek değerler atanır. Örneğin, `Main()` metodu, onu `Hesapla(4,8)` diye çağırmaktadır. Bu durumda `a=4` ve `b=8` ataması yapılıyor demektir. Bazan metodun parametresi olmaz. Parametreleri olmasa bile metot adının sonuna boş () parantezlerini koymak zorunludur. Metot adının önünde `int` yazılıdır. Bu metodun alacağı değerin `int` tipinden olacağını belirtir. Her metot bir değer almak zorundadır. Bu değer herhangi bir veri tipi ya da `void` (boş küme) olabilir.

```
int Hesapla(int a, int b)
```

ifadesi metodun adını, parametrelerini ve değer kümesini (alacağı değerin veri tipi) belirten bir ifadedir. Metoda ait gerekli her şeyi belirttiği için, bu ifadeye *metodun imzası* ya da *metodun belirtkesi* denilir.

Metot imzasını önündeki `static` nitelemesinin ne işe yaradığını yukarıda söylemiştik.

Metodun imzasından sonra gelen `{}` bloku *metot bloku* ya da *metot gövdesi* adını alır. Metoda ait değişkenler ve deyimler burada yer alır. Metot gövdesindeki

```
int x, y;
```

bildirimi x ve y adlı `int` tipinden iki değişken bildiriyor. Metot gövdesinde yer alan değişkenlere metodun yerel değişkenleri denilir. Yerel değişkenlere ancak metot içinden erişilir, dışarıdan erişilemez.

```
x = 3 * a;
y = b / 2;
```

deyimlerinin birincisi, metot çağrılırken `a` ya atanan değerin 3 katını `x` değişkenine atıyor. İkincisi ise, metot çağrılırken `b` ye atanan değerin yarısını `y` değişkenine atıyor.

```
return x + y;
```

deyimi yukarıda bulunan x ve y değerlerini toplayıp `Hesapla()` metodunun değeri olarak belirliyor. Dolayısıyla, `Hesapla(4,8)` çağrısında metodun değeri 16 olmaktadır. Değer kümesi void (boş küme) olmayan her metodun son deyimi `return` anahtar sözcüğü ile başlayıp değeri belirten bir ifade olmak zorundadır.

Bir programda herhangi bir sınıf içinde yer alan değişken ve metotlar kendi başlarına bir iş yapamazlar. Onların program içinde çağrılmaları gerekir. Bizim bu derste ağırlıklı olarak ele alacağımız konsol uygulamalarında, programın *giriş yeri* daima bir `Main()` metodu olacaktır. Dolayısıyla, her programın bir `Main()` metodu olacaktır. Main() metodu kendi sınıfında veya başka sınıflardaki değişken ve metotları çağırıp kullanabilir. Tabii, çağrıya olumlu yanıt verilebilmesi için, çağrılan öğenin erişim belirtecinin yeterli olması gerekir. *Erişim belirteçleri* 'ni ileride ele alacağız.

`Main()` metodunun gövdesindeki deyimlere bakalım.

```
    {
        int n;
        n = Hesapla(4, 8);
        sonuç = n;
        Console.WriteLine(sonuç);
    }
```

Birinci deyim, `int` tipinden `n` adlı bir yerel değişken tanımlıyor.

```
n = Hesapla(4, 8);
```

deyiminde eşitliğin sağ yanı, `Hesapla()` metodunu, parametrelerine `a=2` ve `b=8` değerlerini koyarak çağırıyor. Çağrılan `Hesapla()` metodu çalışıyor ve gövdesinde belirtilen işlemleri yaptıktan sonra bulduğu `x+y` sayısını `return` anahtar szcüğü ile değer olarak alıyor. Sonuçta, `Hesapla(4,8) = 16` değeri çıkıyor. Son olarak `n = Hesapla(4,8)` atama deyimi `n=16` atamasına denk bir iş yapmış oluyor.

Üçüncü satırdaki `sonuç = n` deyimi `n` yerel değişkeninin değerini `sonuç` adlı sınıf değişkenine aktarıyor. Son olarak,

```
Console.WriteLine(sonuç);
```

deyimi sonuç değişkenine atanan 16 değerini konsola yazıyor.

Write() ve WriteLine() Metotlarının Başka Hünerleri

Şimdiye kadar C# 'a bir şeyler yazdırmak için *System.Console* sınıfına ait Write() ve WriteLine() metotlarını kullandık. Şimdi onların yeni hünerlerini göreceğiz.

Program008.cs

```
class Program008
{
      static void Main()
            System.Console.WriteLine("Merhaba, ");
            System.Console.WriteLine("C# dersine hoş geldiniz. ");
}
```

Bu programın çıktısı

```
Merhaba,
C# dersine hoş geldiniz.
```

biçiminde iki satırdan oluşur. Şimdi 4-üncü satırdaki WriteLine() yerine Write() metodunu yazalım:

Program008a.cs

```
class Program008a
{
      static void Main()
      {
            System.Console.Write("Merhaba, ");
            System.Console.WriteLine("C# dersine hoş geldiniz.");
      }
}
```

Bu programın çıktısı

```
Merhaba, C# dersine hoş geldiniz.
```

biçiminde tek satırdan oluşur. Bu ikisinden, daha önce de söylediğimiz bir ipucu çıkarabiliriz.

İpucu

`Write()` metodu parametrelerin değerlerini ekrana yazar ve satırbaşı yapmaz. `WriteLine()` metodu ise parametrenin değerlerini ekrana yazar ve satırbaşı yapar.

Yukarıdaki programlarımız her koşmada ekrana bir tek ileti yazdılar; çünkü herbirinde WriteLine() metodunun bir tane parametresi vardı. Dolayısıyla, ekrana o parametrelerin değerleri yazıldı. Acaba programımız bir

koşmasında birden çok iletiyi yazamaz mı? Tahmin edeceğiniz gibi bu sorunun yanıtı ‘evet’ dir. Öyle olduğunu aşağıdaki örneklerden görebiliriz.

En kolay görünen ama en çok kaçınmamız gereken yöntem, her parametre için bir tane Write() ya da WriteLine() metodu kullanmaktır:

Program008b.cs

```
class Program008b
{
    static void Main()
    {
          int puan;
          puan = 19;
          System.Console.WriteLine(puan - 2);
          System.Console.WriteLine(puan * 2);
          System.Console.WriteLine(puan / 2);
          System.Console.WriteLine(puan % 2);
          System.Console.WriteLine(123 + 456);
          System.Console.WriteLine(12*2 + 2 - 8/4);
    }
}
```

Bu programın çıktısının

```
17
38
9
1
579
24
```

olduğunu göreceksiniz. Şimdi parametrelerin değerlerini aynı satıra yazdırmak için `WriteLine()` yerine `Write()` metodunu kullanalım.

Program008c.cs

```
class Program008c
{
    static void Main()
    {
        int puan;
        puan = 19;
        System.Console.Write(puan - 2);
        System.Console.Write(puan * 2);
        System.Console.Write(puan / 2);
        System.Console.Write(puan % 2);
        System.Console.Write(123 + 456);
        System.Console.Write(12 * 2 + 2 - 8 / 4);
```

```
    }
}
```

Bu programın çıktısı

```
17389157924
```

olur. Bu çıktı özünde doğru olmakla birlikte, parametrelerin değerleri bitişik yazıldığı için, okuyan birisinin çıktıyı doğru algılaması olanaksızdır. Bu sorunu çözmek için şu yöntemi izleyeceğiz:

Yer Tutucu Operatörü {}

{} parantezinin blokları belirlediğini söylemiştik. Şimdi onun başka bir işlevini göreceğiz. Önce aşağıdaki programın çoktılarına bakalım.

Program009.cs

```
class Program008d
{
    static void Main()
    {
        int puan;
        puan = 19;
        System.Console.Write("{0}  " , puan - 2);
        System.Console.Write("{0}  " , puan * 2);
        System.Console.Write("{0}  " , puan / 2);
        System.Console.Write("{0}  " , puan % 2);
        System.Console.Write("{0}  " , 123 + 456);
        System.Console.Write("{0}  " , 12*2 + 2 - 8/4);
    }
}
```

Bu programın çıktısı

```
17  38  9  1  579  24
```

olur. Bu çıktı özünde doğru olduğu gibi, okuyanı da yanıltmaz. ("{0} " , puan – 2) ifadesinde " " içindekiler bir metin olarak konsola yazılır. Bu metin içinde {0} simgesi yer tutucu operatördür. Metinden sonraki ilk değişkenin yazılacağı yeri gösterir. Yer tutucu operatörünün bir çok hünerini ilerleyen derslerde aşama aşama öğreneceğiz.

Şimdiye dek Write() ve WrireLine() metotlarının çıktıları tek bir parametre değeri oldu. Acaba birden çok parametre olunca, istenen parametrelerin değerleri yazdırılamaz mı? Bu sorunun yanıtının *'evet'* olduğunu aşağıdaki program göstermektedir.

Program0010.cs

```
class Program010
{
    static void Main()
    {
```

```
            System.Console.WriteLine("{0} {1}", 25 , 35 , "Merhaba");
      }
}
```

Bu programın çıktısı

```
25      35
```

biçiminde ilk iki parametreden oluşur. Bu ikisinden bir ipucu çıkarabiliriz.

İpucu

Parametreler soldan sağa doğru 0,1,2,... biçiminde numaralanır. Programlama dillerinin çoğu numaralama işlemini 1 den değil 0 dan başlatır. Biz de paramatreleri, soldan sağa doğru yazış sıramızla 0 dan başlayarak numaralayacağız. Buna göre,

```
 "{0}"  ilk parametreyi (0-ıncı) yazar. Örneğimizde 0-ıncı parametre 25 dir.
 "{1}"  1-inci parametreyi yazar. Örneğimizde 1-inci parametre 35 dir.
 "{2}"  2-inci parametreyi yazar. Örneğimizde 2-inci parametre "Merhaba" metnidir.
 ...
 "{n}"  n-inci parametreyi yazar.
```

Parametre değerlerinden istediklerimizi istediğimiz sırada yazdırabiliriz. Örneğin, yukarıdaki programda `{2}` yer tutucusu olmadığı için "Merhaba" parametresi çıktıya gitmemiştir. Aşağıdaki program önce 2-inci sonra 0-ıncı parametreleri yazdırır.

Program011.cs

```
class Program011
{
      static void Main()
      {
            System.Console.WriteLine("{2} {0} " , 25 , 35 , "Merhaba");
      }
}
```

Bu programın çıktısı şöyle olacaktır:

```
Merhaba 25
```

biçiminde olacaktır.

Write() ve WriteLine() metotlarımız her türlü aritmetik işlemi yapma becerisine sahiptirler. Aşağıdakine benzer programlar yazıp, sonuçlarını görünüz.

Program012.cs

```
class Program012
{
    static void Main()
    {
        System.Console.WriteLine(12 * 12 + 35 - 412);
    }
}
```

Değişken Kullanımına Örnekler

Değişkene atanan ilk değeri, program koşarken değiştirebiliriz. Bunu görmek için, programımızı aşağıdaki gibi değiştirelim.

Program013.cs

```
class Program013
{
    static void Main()
    {
        int puan = 19;
        System.Console.WriteLine(puan);
        puan = puan + 4;
        System.Console.WriteLine(puan);
    }
}
```

Bu programın çıktısı

```
19
23
```

olur. Neden böyle olduğunu kolayca anlayabilirsiniz. Kaynak programın 5-inci satırında puan değişkenine ilk değer olarak 19 atanmıştır. Bu değer geçerli iken 6-ıncı satırdaki WriteLine() metodu bu ilk değeri yazar. Ama arkasından gelen 7-inci satırdaki

```
puan = puan + 4 ;
```

deyimi, puan değişkeninin değerini 4 artırarak 23 yapmıştır. Dolayısıyla , 8-inci satırdaki WriteLine() metodumuz, o anda puan 'ın değeri olan 23 sayısını yazacaktır.

Şimdi, uzaylı dostumuz kulağımıza bir ipucu fısıldamaktadır.

İpucu

Yukarıdaki programın 8-inci satırındaki

```
puan = puan + 4 ;
```

atama deyimi

```
puan = 23;
```

atama deyimine denktir. Okuldaki matematik derslerinde yaptığımız işlemlere biraz aykırı görünen bu atama yöntemi, bilgisayar programlarında geçerlidir ve çok işe yarar. Özelikle, program koşarken bir değişkenin değeri, akışta oluşan koşullara göre değişebilir ve o değişimleri kaynak programda yazma olanağı olmayabilir.

Geçerlik Bölgesi (scope)

Şimdi değişkenlerin hangi bloklarda geçerli olabileceklerini araştıralım. Bunun için, puan değişkenini `Main()` blokundan alıp üst bloka koyalım. Programımız aşağıdaki biçimi alsın.

Program014.cs

```
class Program014
{
    int puan = 19;
    static void Main()
    {
        System.Console.WriteLine(puan);
    }
}
```

Yaptığımız değişiklik şudur: `Main()` blokundaki değişken tanımını ve değer atamayı, dış bloka aldık. `WriteLine()` metodu ise olduğu yerde, yani iç blokta kaldı. Bu programı derlemek istediğimizde, derleyici bize aşağıdaki iletiyi gönderir:

```
Program009.cs(6,28): error CS0120: An object reference is reququired for the static field, method, or property 'Program14.puan'
```

Bu iletiden anlamamız gereken şey, 6-ıncı satırda WriteLine() metodunun, parametre olarak yazdığımız `puan` değişkeninin değerine erişemediğidir. Çünkü, değişken adresini işaret eden bir işaretçi (referans, pointer) yoktur. Bu sorunu aşmak için, değişkenimize `static` nitelemesini vermek yetecektir. Neden böyle olduğunu ileride açıklayacağız.

Program014a.cs

```
class Program014a
{
    static int puan = 19;
    static void Main()
    {
        System.Console.WriteLine(puan);
    }
}
```

Bu programı koşturduğumuzda, çıktının

```
19
```

olduğunu göreceğiz. Demek ki, iç bloktaki WriteLine() metodu dış bloktaki değişkeni görebildi ve onun değerini yazdı.

Şimdi aynı değişkeni hem iç, hem de dış blokta tanımlayalım.

Program015.cs

```
class Program015
{
    static int puan = 19;
    static void Main()
    {
        int puan = 25;
        System.Console.WriteLine(puan);
    }
}
```

Programı derlediğimizde, derleyici aşağıdaki uyarıyı iletir, ama programı derler ve yürütülebilir dosyayı yaratır.

```
Program011.cs(3,14): warning CS0169: The private field 'Program015.puan' is never used
```

Gerçekten dir komutu ile Program015.exe dosyasının yaratıldığını görebilir ve

```
Program015
```

yazarak programı koşturabilirsiniz. Programın çıktısı

```
25
```

dir. Şimdi bunu biraz irdeleyelim. Dış blokta ve iç blokta ayrı ayrı iki tane puan değişkeni tanımladık. Dış blokta puan değişkenine 19, iç blokta ise 25 atadık. Program koştuğunda, WriteLine() metodu kendi bloku içindeki 25 değerini yazdı. Dıştaki değeri ihmal etti. Oysa, önceki programda, dış bloktaki değeri yazmıştı. Çünkü iç blokta aynı adlı bir değişken yoktu.

Bu noktada uzaylı dostumuza bakıyoruz. O bize şu açıklamayı yapıyor.

İpucu

İç blokta dış bloktaki değişken adları kullanılabilir. Ancak, aynı adı taşıdıklarında, iç bloktaki değişkenler öncelik alır.

Şimdi de olguyu tersine çevirelim. İç blokta tanımlı değişkenlerin dış blokta kullanılıp kullanılamayacağını öğrenmeye çalışalım. Bunun için `Main()` bloku içine bir blok koyalım ve değişkenimizi orada tanımlayalım.

Program016.cs

```
class Program016
{
    static void Main()
    {
        System.Console.WriteLine(puan);
        {
            int puan = 19;
        }
    }
}
```

Bu programı derlemeye kalkınca, aşağıdaki hata iletisini alırız.

```
error CS0103: The name 'puan' does not exist in the class or namespace 'Program016'
```

Bu ileti, 5-inci satırdaki `WriteLine()` metodunun puan değişkeninin tanımlandığı iç bloku göremediği anlamına gelir.

Şimdi uzaylı dostumuza danışmadan, kendimiz bir ipucu yazabiliriz.

Bazı pencereler dışarıdan bakılınca içerinin görünmemesi için özel bir maddeyle kaplanır. Dışarıdan bakınca camın içerisi görünmez, ama içeriden dışarısı olduğu gibi görünür. Bizim yazdığımız iç-içe bloklar bu özeliğe sahiptirler.

İpucu

İç-içe bloklar olduğunda, iç bloktan dış blok görünür, ama dış bloktan iç blok görünmez. Bu demektir ki, dış bloktaki değişkenlere iç bloktan erişilebilir, ama dış bloktan iç bloktaki değişkenlere erişilemez.

Dizi (array)

Dizi (array) kavramı hemen her dilde var olan önemli bir yapıdır. C# dilinde System.Array sınıfı arraylerle yapılabilecek bütün işleri belirler. Onu ayrı bir bölüm olarak ileride ele alacağız. Ancak o zamana kadar, yeri geldikçe diziler kullanmamız gerekecektir. O nedenle, bize oraya kadar yetecek basit bilgileri şimdi edinmemiz gerekiyor.

Array aynı veri tipinden çok sayıda değişkeni kolayca tanımlamaya ve o değişkenlere kolayca erişmeye olanak sağlayan bir yapıdır. Sözdizimi şöyledir:

```
tur[] ad =  {bileşen değerleri};
```

Örneğin,

```
int[]  puan = {68, 85, 45, 98, 74};                    (1)
```

bildirimi `int` türünden

```
puan[0], puan[1], puan[2],puan[3], puan[4]              (2)
```

`puan` adlı 5 tane değişken yaratır. Bunlara `puan` adlı array'in bileşenleri ya da öğeleri denilir. C# array'in bileşenlerini 0 dan başlayarak sırayla numaralar. (1) bildirimini şöyle de yapabiliriz:

```
int[]  puan  =   new int[5] ;
```

bildirimi puan adlı 5 bileşenli bir array yaratır. Bu arrayin bileleşenlerine değer atamak için, diğer değişkenler için yaptığımız atama deyimini kullanırız:

```
puan[0] = 68;
puan[1] = 85;
puan[2] = 45;
puan[3] = 98;
puan[4] = 74;
```

Array bildiriminde [ ] icindeki i = 0, 1, 2, 3, 4 sayılarına array'in indisleri (damga) denilir. Array içindeki her öğe kendisine ayrılan damga ile kesinlikle belirli bir değişkendir. Array sınıfının `Length` gendeğeri (property) arrayin uzunluğunu; yani kaç bileşeni (öğesi) olduğunu belirtir. Sözdizimi şöyledir:

```
puan.Length
```

Bunun değeri `puan` arrayimiz için 5 olur.

Array ile ilgili olarak söylediklerimiz şimdilik bize yetecektir.

For Döngüsü

Döngüler her programlama dilinde var olan önemli yapılardır. Onları ileride ayrı bir bölüm olarak ele alana kadar gerektiği yerde basit for döngüleri yazacağız. O nedenle, for döngüsünü basitçe açıklamamız gerekiyor.

Bir eylemin birden çok tekrar etmesini istiyorsak, o eylemi yaptıran kodu ard arda istenen sayıda yazabiliriz. Örneğin, yukarıdaki puan arrayinin bileşenlerinin sayısının 5 yerine 500 olduğunu düşünelim ve o bileşen değerlerini alt alta yazdırmak isteyelim. O zaman, `i` indisini `0` dan `499` 'a kadar değiştirerek

```
Console.WriteLine(puan[i]);
```

deyimini 500 kez yazmayı düşünebiliriz. Elbette, bu akıllıca bir yol değildir. Onun yerine, for döngüsü denilen şu yapıyı kullanırız:

```
for (int i = 0 ; i <500 ; i++)
{
    Console.WriteLine(puan[i]) ;
}
```

Bu döngüdeki kodları çözümleyelim:

`for` : döngünün başladığını bildirir.

(int i = 0 ; i <500 ; i++) : döngü sayacının int tipinden `i` değişkeni olduğunu ve ilk değerinin `0` dan başladığını; her adımda `1` artarak `500` den küçük kaldığı sürece `{}` döngü bloku içindeki deyim(ler)in tekrarlanacağını bildirir.

Bu söylediklerimizi bir araya getirerek aşağıdaki programı yazabiliriz.

ArrayYaz.cs

```
using System;

namespace BasitArray
{
    class birArray
    {
        static void Main(string[] args)
        {
            int[] puan;
            puan = new int[5];
            puan[0] = 68;
```

```
            puan[1] = 85;
            puan[2] = 45;
            puan[3] = 98;
            puan[4] = 74;

            for (int i = 0; i < puan.Length ; i++)
            {
                Console.WriteLine("puan[{0}] = {1}", i , puan[i]);
            }
        }
    }
}
```

Bilgisayarda Bellek

Bilgisayarda çalışan bir programın kullandığı aduzayları, nesneler, değişkenler, deyimler, metotlar vb bilgisayarın belleğinde tutulur. Fiziksel yapılarına göre üç tür bellek vardır. Merkezi İşlem Birimi (CPU) içindeki register'ler, ana bellek (RAM) ve kayıt ortamları (hard disk, disk vb).

Registerler

Erişilmesi en hızlı olan bellek bölgeleridir; ama çok kısıtlıdırlar; CPU'nun mimarisine göre değişirler. Daha önemlisi, programcı register'lerin nasıl kullanılacağına karar veremez. Onları derleyici yönetir. Çok kullanılanları register'lerde tutarak hız kazanır.

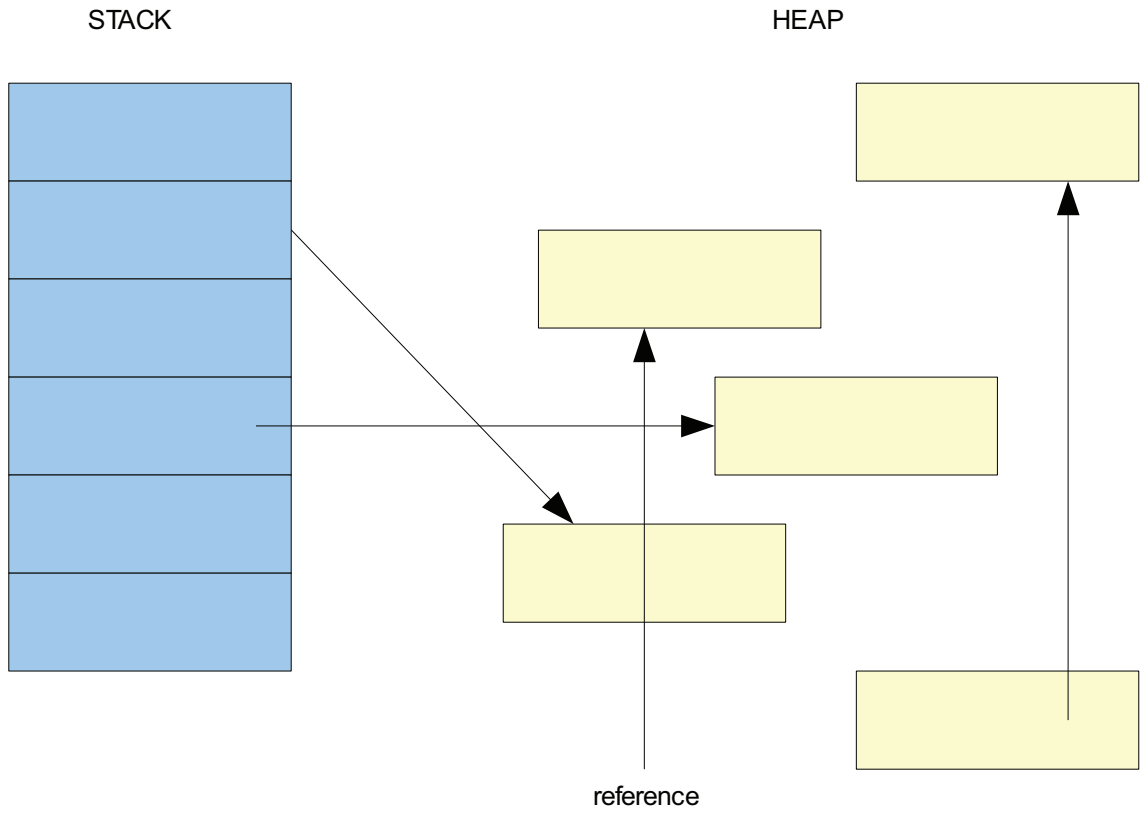
Kayıt ortamları

Program ana bellekten silindikten sonra gerektiğinde tekrar ulaşmak istediğimiz veriler, programlar ve benzerlerinin kalıcı olarak kaydedildiği ortamlardır. Hard disk, floppy disk, teyp, CD benzeri çok çeşidi vardır. Ayrıca, büyük programlar koşarken, ana bellek yetmediği zaman derleyici hard diskin bir bölümünü yardımcı bellek olarak kullanabilir. Swap space denilen bu alanı yönetmek derleyicinin ve sistemin tekelindedir.

Ana Bellek (RAM)

Program koşarken programın öğeleri ana bellekte belirli yerlerde tutulur. Çok ayrıntıya girmeden iki önemli bölgeyi söylemek gerekir: Stack ve Heap .

Stack : LIFO (Last Input First Output – son giren önce çıkar) adıyla bilinen yapıdır. Üst üste yığılmış kutuları andırır. Yeni gelen kutu en üste yerleşir, alınacak kutu en üstten alınır. RAM'de en hızlı erişim sağlanan bölgedir. C# dilinde bütün değer tipleri stack 'ta tutulur. `Stack` bir sınıftır ve stack ile ilgili bütün işleri yapmaya yetecek öğelere sahiptir.



Heap: C# dilinde bütün referans tiplerinin tutulduğu bellek bölgesidir; dolayısıyla nesneler `heap`'te tutulur. Bir sınıfa ait bir nesne yaratılınca, `Heap` içinde nesnenin bütün öğelerini içerecek bir bellek bloku ayrılır. Burası nesnenin adresidir. Bu adres, bu nesneyi işaret eden *referans* (işaretçi) tarafından bilinir. Şekildeki oklar referansları (pointer, işaretçi) göstermektedir.

Çöp Toplayıcı (Garbage Collector)

Java dilinde olduğu gibi, C# bir sınıftan yaratılan bir nesnenin işi bitince, ana bellekte ona ayrılan yeri boşaltır ve heap 'a ekler. Bu işin ayrıntısına girmeden, basit bir açıklama yapabiliriz. Bir nesne yaratılınca onu işaret eden bir referans (işaretçi) vardır. Örneğin,

```
{
string str = new string("Bu gün hava güzeldir.");
}
```

nesne kurucu deyimi ile yaratılan "Bu gün hava güzeldir." nesnesi ana bellekte bir adrese yerleşir. O adresi işaret eden referans `str` dir. Program kontrolu nesneyi içeren { } blokunu geçtiğinde `str` referansı yok olur. Ama "Bu gün hava güzeldir." nesnesi bellekteki yerini korumaya devam eder. C++ dilinde, işi biten nesneyi bellekten silmek, programcının görevidir. Programcı bunu doğru yapmadığı zaman, bellekte kendilerine asla erişilemeyen nesneler yer alır. Bu olgu C++ dilinin en büyük kusuru sayılır. Java ve ondan sonra gelen nesne yönelimli diller, Çöp Toplayıcı adını alan bir yöntemle, bellekte işleri bittiği için işaret edilmeden kalan nesneleri toplar ve çöpe atar; yani onların işgal ettikleri bellek adreslerini boşaltır ve o adresleri heap 'a ekler.

Dolayısıyla, C# programcısının, işi biten nesneyi silmek gibi bir yükümlülüğü yoktur; o işi Çöp Toplayıcı kendiliğinden yapacaktır. Ancak ileri düzeydeki programcılar, C++ dilindeki gibi pointer kullanmak, yarattıkları nesneleri işi bitince yoketmek isteyebilirler. C# buna kısıtlı izin verir. Ama bu konu, bu kitabın kapsamı dışındadır.

Alıştırmalar

1. Aşağıdaki program konsoldan girilen bir satırı string olarak okur ve onu konsola yazar.

```
using System;
namespace Bölüm01
{
    class InputOutput01
    {
        public static void Main()
        {
            string s;
            s = Console.ReadLine();
            Console.WriteLine(s);

        }
    }
}
```

Çıktı

```
Ankara başkenttir.
Ankara başkenttir.
Devam etmek için bir tuşa basın . . .
```

2. Aşağıdaki program konsoldan girilen karakterlerden yalnızca ilkini okur ve onu konsola yazar.

```
using System;
namespace Bölüm01
{
    class InputOutput02
    {
        public static void Main()
        {
            char ch;
            ch = (char)Console.Read();
            Console.WriteLine(ch);
        }
    }
}
```

Çıktı

```
765
7
```

Kullanıcı klavyeden 765 girdiği halde `Read()` metodu yalnızca ilk karakter olan 7 karakterini okumuştur. Burada şuna dikkat etmeliyiz. `Read()` metodu okuduğu karakteri, bizim gördüğümüz harf veya rakam biçimiyle değil, o karakterin `ASCII` koduyla algılar. Başka bir deyişle `Read()` metodu 7 karakterini okuduğunda buffer'a aldığı değer 7 değil, 7 rakamının ASCII kodu olan 55 dir. Dolayısıyla bu kodu 7 karakterine dönüştürmek için

```
ch = (char)Console.Read();
```

kapalı (implicit) dönüşüm operatörünü (char) kullanıyoruz. Bunu daha iyi anlamak için, yukarıdaki programı biraz değiştirelim.

Aşağıdaki program derlenemez. Çünkü, yalnızca sayısal veri tipleri arasında tip dönüşümü yapılabilir.

```
using System;
namespace Bölüm01
{
    class InputOutput03
    {
        public static void Main()
        {
            int ch;
            ch = (char)(Console.ReadLine());
            Console.WriteLine(ch);
        }
    }
}
```

Çıktı

```
Error   1       Cannot convert type 'string' to 'char' ...
```

Çıktıdan görüldüğü gibi, string'den char tipine açık (explicit) dönüşüm yapılamamaktadır.

3. Aşağıdaki program konsoldan girilen karakterlerden yalnızca ilkini okur ama (char) dönüşümü istenmediği için onu karaktere dönüştürüp konsola yazamaz.

```
using System;
namespace Bölüm01
{
    class InputOutput01
    {
        public static void Main()
        {
            char ch;
            ch = Console.Read();
            Console.WriteLine(ch);
        }
    }
}
```

Bu program sözdizimi (syntax) açısından yanlıştır; derleyici programı derlemez ve şu hata iletisini verir:

```
Error   1       Cannot implicitly convert type 'int' to 'char'. An explicit conversion exists (are
you missing a cast?)
```

4. Aşağıdaki program konsoldan girilen karakterlerden yalnızca ilk üçünü sırayla okur ve ekrana yazar.

```
using System;
namespace Bölüm01
{
    class InputOutput01
    {
```

```
        public static void Main()
        {
            char c1, c2, c3;
            c1 = (char)Console.Read();
            c2 = (char)Console.Read();
            c3 = (char)Console.Read();
            Console.WriteLine(c1);
            Console.WriteLine(c2);
            Console.WriteLine(c3);

        }
    }
}
```

Çıktı

```
a1b2c3d4e5f6
a
1
b
```

Bölüm 02

Visual Studio Ortamı

Visual Studio

.NET Framework

Visual C# Arayüzü ile Program Yazma, Derleme ve Koşturma

Sıkça Sorulan Sorular

Visual Studio

Birinci Bölümde, kaynak programı, derlemeyi, gerektiğinde programı düzeltmeyi (debug) ve derlenen programı koşturmayı (run) öğrendik. Bu işleri yaparken basit bir editör ile C# derleyicisine (cs.exe) gereksememiz oldu. Programı derleme ve koşturma eylemlerini konsoldan yazdığımız komutlarla gerçekleştirdik. Hepsi çok yalın ve kolay işlerdi. O yaptıklarımız hemen her dildeki program için yapılanlardır ve işin özüdür. Programcılığa başlayan herkes o süreci bilmek zorundadır.

Öte yandan, o yalın eylemler, gerekseme duyduğumuz tekerleği kendi ellerimizle yapmak gibidir. Her zaman mümkündür; ama en iyi yol değildir. Birileri bizim için tekerleği yapmışlarsa, onu hazır alıp arabamızı yürütmek daha akıllıca olabilir.

Günümüzde programlama işi biraz tekerlek yapmaya benzedi. Mükemmel hazır tekerleri alıp kullanıyormuşçasına kaynak programı yazmayı, derlemeyi ve koşturmayı çok çok kolaylaştıran araçları alıp kullanabiliriz. Bu tür araçlara bütünleşik (integrated) program, kullanıcı dostu program, arayüz gibi adlar verilir. Hemen her dilde bu tür araçlar bolca üretilmiştir ve programcının emrindedir.

C# için de programlamanın evrelerini kolaylaştıran araçlar vardır. Biz, bu derste Microsoft'un geliştirdiği *Visual Studio 2008* bütünleşik programını (arayüz) kullanacağız. Bunun iki sürümü vardır. Birisi profesyonel programcılar için lisanslı olan sürümüdür. Ötekisi, programlamaya yeni başlayanların bütün gereksemelerini karşılayacak nitelikteki serbest sürümüdür. Ücretsizdir, lisans almaya gerek yoktur ve internetten indirilebilmektedir.

Visual Studio 2008, Microsoft'un C++, C#, VB, J# , Jscript dillerinde program yazmayı ve derlemeyi kolaylaştırmak için hazırladığı çok amaçlı bir arayüzdür.

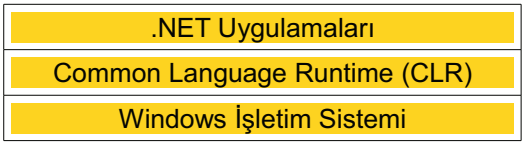
.NET Framework

Programlamaya yeni başlayanları gereksiz ayrıntıyla boğmamak için, *Visual Studio*'nun iç yapısını açıklamaya girmeyeceğiz. Onu bir otomobil gibi görelim, motorunun kaputunu hiç açmadan direksiyonuna oturalım ve kumanda tablosundaki düğmelerin işlevlerini öğrenelim. Çünkü biz başlangıçta motor yapımcısı değil, araba sürücüsü olmak istiyoruz. Sürücü belgesi bize uzun zaman yetecektir. Epeyce zaman sonra zaten merak ederek motorun kaputunu açıp, onun içindekileri görmek isteyeceğiz. Ancak, şu kadarını söylemek hem yararlı hem gereklidir. Microsoft *C++, C#, VB, J#* gibi dilleri derlemek ve çalıştırmak için *.NET Framework* denilen ortak bir platform yarattı. C# kaynak programları kendiliğinden o ortama taşınır, orada derlenirler, orada çalışırlar. O nedenle, C# dilini öğrenirken, ayrı bir çaba harcamadan ve çoğunlukla farkında bile olmadan *.NET Framework* yapısını da epeyce öğrenmiş olacağız.

.Net Framework genel amaçlı bir program geliştirme platformudur. 2000 yılında Microsoft firması tarafından ilk sürümü ortaya konmuştur. Bu kitap yazılırken, 3.5 sürümü piyasaya çıkmış durumdadır. Zamanla yeni sürümlerinin çıkması doğaldır. İşlevleri bakımından *java platformu* ile aynıdır. Her ikisiyle istemcide, sunucuda, mobil ya da gömülü cihazlarda uygulama programları geliştirilebilir, web uygulamaları, veritabanı uygulamaları gibi bir çok iş yapılabilir. Her ikisi de işletim sistemine ve donanıma bağlılığı ortadan kaldırmayı hedeflemektedir.

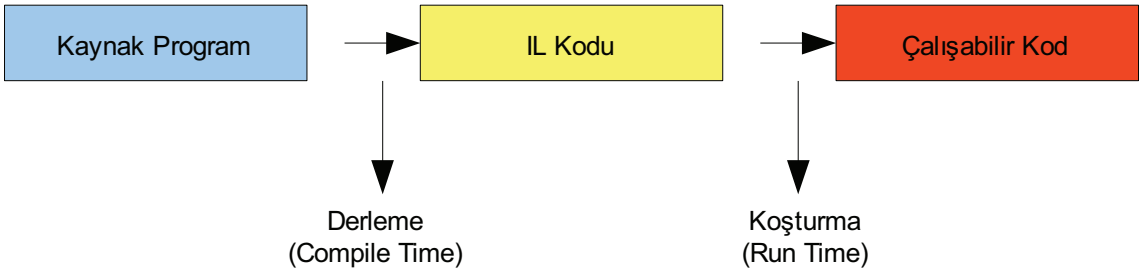
Bunu yapabilmek için, java, bellekte *virtual machine (VM)* denilen sanal bir makina yaratır. Java derleyicisi kaynak programı *bytecode* denilen ve *VM* üzerinde çalışabilen bir tür sanal makina diline dönüştürür. Böylece platformdan bağımsızlık sağlanır.

Microsoft'un uygulama geliştirme dilleri olan *C++, C#, VB, J#* ile yazılan kaynak kodlar derlenince *IL (Intermediate Language)* koduna dönüşürler. Bu dönüşüm, java'nın *bytecode*'una benzetilebilir. Sonra *JIT (Just-in-Time)* derleyicisi koşturulacak *IL* kodunu makine diline dönüştürür. *IL* bazı kaynaklarda *MSIL (Microsoft Intermediate Language) veya CIL (Common Intermediate Language) diye adlandırılır. CLR (Common Language Runtime)* ise *IL* koduna dönüşen programların koştuğu yerdir.



Başka bir deyişle, *C++, C#, VB, J#* dillerinden biriyle yazılan bir program derlenince doğrudan *Windows İşletim Sistemi* üzerinde çalışmaz; onun üzerinde kurulan sanal *CLR* üzerinde çalışır. Çünkü *C++, C#, VB, J#* dillerinde yazılan programlar derlenince, Windows İşletim sisteminde çalışabilen (executable) kodlara değil, *IL* diline dönüşür. Bunun ne anlama geldiğini, basite indirerek, şöyle açıklayabiliriz. C# dilinde `int` tipi bir değişken tanımladığımızda, derleyici onu *.NET Frameworkta'ki* karşılığı olan `Int32` tipine dönüştürür. Bunun gibi, kaynak programı hangi dilde yazarsanız yazınız, oradaki her şey *.NET Framework'taki* karşılığına dönüşür. Genelleştirirsek şöyle diyebiliriz. *C++, C#, VB, J#* dillerinin veri tipleri .NET'in *CTS* (Common Type System) denilen veri tiplerine dönüşür. Böylece bu diller için ortak bir platform yaratılmış olmaktadır. C# dilinin veri tiplerinin .NET 'deki karşılıklarını *Veri Tipleri ve Değişkenler* bölümünde ayrıntılı olarak ele alacağız.

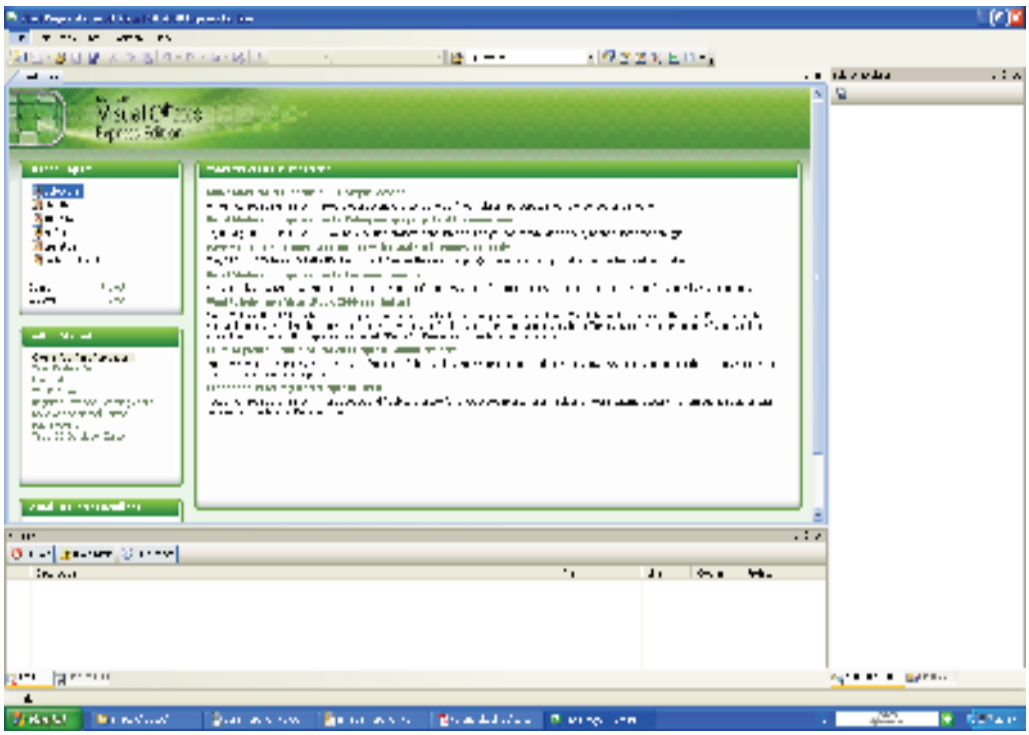
.NET Framework çok geniş bir kütüphaneye sahiptir. Onunla ilişkili olarak *CLI* (Common Language Infrastructure), *CTS* (Common Type System), metadata, *VES* (Virtual Execution Environment), *CLS* (Common Language Specification) gibi kavramları duyabilirsiniz. Ama onları şimdilik gözardı edebilirsiniz. Çünkü biz, Visual Studio ve .NET 'in iç yapısıyla değil, kumanda odasıyla ilgili olacağız.



Bu kısa açıklamadan sonra, Visual Studio ile çalışmaya başlayabiliriz. Bilgisayarınızda *Visual Studio 2008* 'in içinde ayrı bir modül olan *Microsoft Visual C# 2008 Express Edition* 'in yüklü ve çalışır durumda olduğunu varsayıyoruz. Nasıl yükleneceğini internetten indirdiğiniz yerden okuyabilirsiniz.

Visual C# Arayüzünü Başlatmak

Başlat **->** *Programlar -> Microsoft Visual C# 2008 Express Edition* sekmesini tıklayın. Karşınıza şuna benzer bir pencere gelecektir.

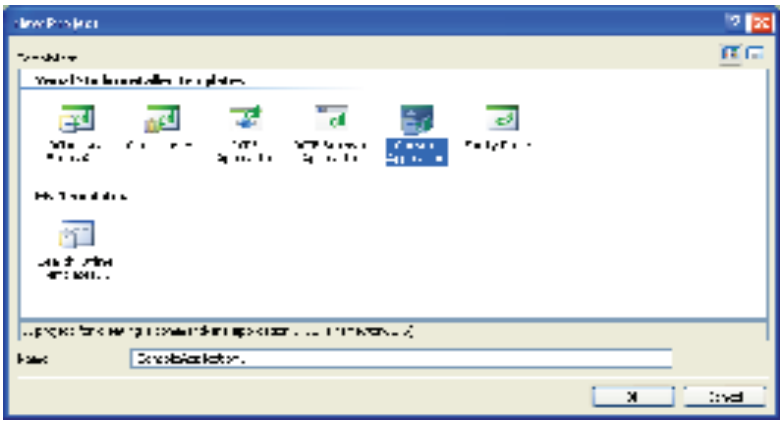


Sol üstteki `Recent Projects` bölümü sizde farklı olacaktır.

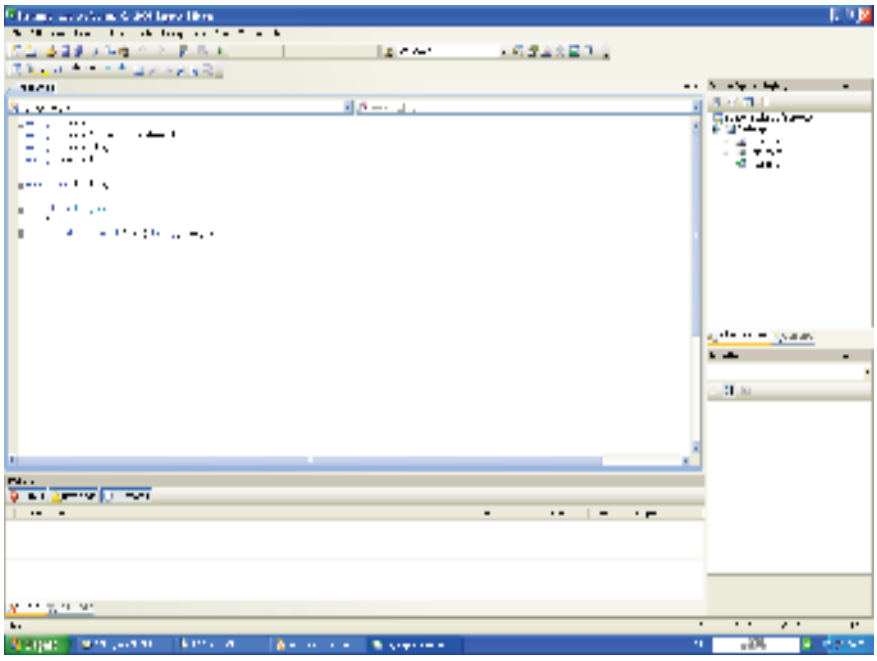
Visual Studio ilk açılışında projeleri kaydedeceği dizini sorar. O anda istediğiniz sürücüde istediğiniz bir dizin yaratabilir veya Visual Studio'nun önerdiği öndeğer (default) proje dizinini kabul edebilirsiniz. Bu

kitabın yazıldığı makinada c:\vsProject adlı bir dizin yaratılmıştır. Bütün projeler oraya kaydedilmektedir.

Üst satırda menü, onun altında araçların yer aldığı satır yer alır. Onun altındaki satırın solunda *Start Page*, sağında *Solution Explorer* sekmelerini göreceksiniz. *Start Page* sekmesine sağ tıklayınız ve açılan pencerede *Close* 'u tıklayınız. Görüntüdeki pencere kapanacaktır. Sonra File -> New Project 'i tıklayınız. Görüntüye *New Project* penceresi gelir:



Bu pencerede bize altı seçenek sunuluyor. İlk işlerimiz konsol uygulamaları olacaktır. C# ile nesne programlamayı onunla öğreneceğiz. Ondan sonra öteki uygulamaları kolayca yapabilir duruma geleceğiz. Pencereden *Console Application* 'i tıklıyoruz. Onun ikonu seçili duruma gelir. Sonra, pencerenin altındaki *Name* alanına gidiyoruz. Bu alan, yapacağımız konsol uygulamasının hangi aduzayına (*namespace)* ait olduğunu belirliyor. *Name* alanında *ConsoleApplication1* adı yazılıdır. Olduğu gibi bırakabiliriz ya da istediğimiz başka bir ad verebiliriz. *Başlangıç* diyelim ve OK düğmesine basalım. Karşımıza şu pencere çıkar:



Sol üstte *Başlangıç.Program* sekmesi altında Visual Studio 'nun editör pencesi görünüyor. Bu pencerede,

kaynak program için Visual Studio bazı temel kodları hazır yazılmış olarak önümüze koyuyor.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Başlangıç
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Bu programın ilk dört satırının her birisindeki *using* sözcüğü bir *namespace (aduzayı)* çağırıyor. İlke olarak, çağrılan bir ad uzayındaki sınıflar, o sınıfların içindeki değişkenler ve metotlar sanki bu programın içinde tanımlanmış gibi kullanılabilir (kısıtları ayrıca göreceğiz). İstediğimiz zaman, `using` anahtar sözcüğü ile başka aduzayları da çağırabiliriz. İlk derslerde, yalnızca *System* aduzayına gereksememiz olacak. O nedenle, sonraki üç satırı silebiliriz. Silmesek de bize bir zarar vermezler.

Visual Studio, bize yaptığı program önerisinde aduzayının (`namespace`) adının '*Başlangıç'* olduğunu belirtiyor. Zaten bu adı biz vermiştik. Şimdi yazacağımız programdaki sınıf veya sınıflar, *Başlangıç* aduzayına ait olacaklardır. Bu aidiyet, fiziksel kayıt ortamında değil, mantıksal (logic) olarak yapılır. Dolayısıyla, biz programlarımızı istediğimiz dizin veya alt dizine kaydedebiliriz.

Hemen belirtelim ki, yazdığımız her program için bir namespace (aduzayı) belirlemek zorunda değiliz. Programa namespace blokunu koymadığımız zaman, yarattığımız sınıflar hiç bir aduzayına ait olmazlar; ama kendi başlarına çalışabilirler; başka aduzaylarını çağırabilirler.

Programın sonraki satırlarına bakarsak şunu görüyoruz. *Visual Studio*, bize sınıf (class) adı olarak *Program* öneriyor ve hemen o sınıfın içinde `Main()` metodunun imzasını atıyor. Sınıf adını aynen bırakabilir veya istediğimiz bir ad ile değiştirebiliriz. Örneğin, *Giriş* diyelim. İlk programımızda *Visual Studio* ile kaynak programımızı yazmayı ve derleyip koşturmayı öğrenmek istiyoruz. Onun için, basit bir deyim yazmakla yetineceğiz. Gelenek olduğu gibi, konsola `"Merhaba C#"` yazdıralım. Bunu yazdırabilirsek, aynı yöntemle bir roman bile yazdırabileceğimizi biliyoruz. Bunu yapmak için, önceki bölümde gördüğümüz gibi, `Main()` metodunun gövdesine `Console.Writeline( "Merhaba C# " )` deyimini eklememiz yetecektir. Böylece, *Visual Studio* 'nun editör penceresinde programımız şu biçimi alır.

```
using System;

namespace Başlangıç
{
    class Giriş
    {
        static void Main(string[] args)
        {
            Console.WriteLine( "Merhaba C# " ) ;
```

```
            }
        }
}
```

Şimdi programımızı derleyebiliriz. Ama ondan önce yazdıklarımızı kaydetmekte yarar var. Ekranın sağında yer alan *Solution Explorer – Başlangıç* penceresindeki *Program.cs* Visual Studio 'nun önerdiği addır. Bu adı koruyabiliriz veya istiyorsak değiştirebiliriz.

C# programları `.cs` uzantısı alır ve sınıfın adını almak zorunda değildir. C# buna aldırmaz. Ama içinde bir tek sınıf olan programlarda sınıf adı ile program adını aynı yaparsak, çağrışımda bize kolaylık sağlar. Nasıl yapıldığını görmek için, *Solution Explorer – Başlangıç* penceresindeki *Program.cs* adına sağ tıklayalım, açılan penceredeki *Rename* sekmesini seçerek, yeni adı girelim. Sınıfın adını taşısın diye, programın adını *Giriş.cs* olarak değiştirelim. İsterseniz başka ad da verebilirsiniz. Sonra *File -> Save Giriş.cs* seçeneğini tıklayarak, programı kaydedelim. Aynı anda birden çok programla çalışabiliriz. Hepsini kaydetmek için *File -> Save All* seçeneğini tıklarız. Aynı işi, araç satırındaki

düğmeleriyle de yapabiliriz. Soldaki düğme, etkin olan penceredekini kaydeder. Sağdaki düğme, bütün programları kaydeder. Artık programımızı derleyelim.

Menü satırında *Debug _> Start Without Debugging* sekmesine tıklayınız. İsterseniz onun yerine `Ctrl+F5` tuşlarına da basabilirsiniz. Konsol açılır ve üzerinde *Merhaba C#* görünür:

Böylece, *Visual Studio* ile ilk programımızı yazdık, derledik ve koşturduk. Bundan sonrakiler de bu kadar basit olacaktır. Şimdi *Visual Studio* 'nun bazı hünerlerini görelim.

En sondaki '}' parantez simgesini kaldırıp programı derlemeyi deneyelim. Derleyiciden şu uyarı mesajını alırız.

Error	Description	File	Line	Column	Project
1	} expected	C:\vsProjects\Başlangıç\Başlangıç\Giriş.cs	11	6	Başlangıç

Program yazdıkça, benzer mesajları sürekli alıyor olacağımız için, bu mesajın nasıl okunacağını bir kez açıklamakta yarar olacaktır:

`Error` sütunu programda derleyicinin bulduğu hataların sayısı verilir. Bizim programda 1 hata olduğu işaret ediliyor.

`Description` sütununda derleyicinin gördüğü hatanın açıklaması verilir. Örneğimizde '`} expected`' dendiğine göre, programımızda ' } ' simgesinin eksik olduğu belirtiliyor.

`File` sütununda hatanın hangi dosyaya ait olduğu belirtilir.

`Line` sütununda, hatanın kaçıncı satırda olduğu belirtilir. Örneğimizde, hatanın 11-inci satırda görüldüğü belirtiliyor.

`Column` sütununda hatanın hangi kolonda oluştuğu belirtililir. Örneğimizde, hatanın 6-ıncı kolonda oluştuğu işaret ediliyor. [Kullandığımız ekranın çözünürlüğüne bağlı olarak, bizim saydığımız kolon numarası, derleyicinin saydığı kolon numarasından farklı olabilir. Ama derleyici, hatayı gördüğü yere küçük kırmızı dalga simgesi koyar. Hatayı orada aramalıyız.]

`Project` sütununda programın hangi aduzayına (namespace) ait olduğu belirtilir. Örneğimizde `C:\vsProjects\Başlangıç\Başlangıç\Giriş.cs` dosyası yazılıdır. Burada yazılanlar soldan sağa doğru şunlardır: dosyanın kaydedildiği sürücü adı (`C:`), dizin adı (`vsProject`), aduzayı (`Başlangıç`), sınıf adı (`Başlangıç`) ve dosya adı (`Giriş.cs`).

Hatanın işaret edildiği yere imleci götürüp '}' simgesini yazalım. Sonra, başka bir hata yapalım. Örneğin, gövde içindeki deyimin sonundaki (;) simgesini kaldıralım. Bu kez derleyicimiz hatanın 10-uncu satır, 45-inci sütunda (;) eksikliğinden kaynaklandığını belirten hata mesajını iletecektir. Ayrıca hatanın oluştuğu yere kırmızı renkli küçük bir dalda simgesi koyacaktır.

Error	Description	File	Line	Column	Project
1	; expected	C:\vsProjects\Başlangıç\Başlangıç\Giriş.cs	10	45	Başlangıç

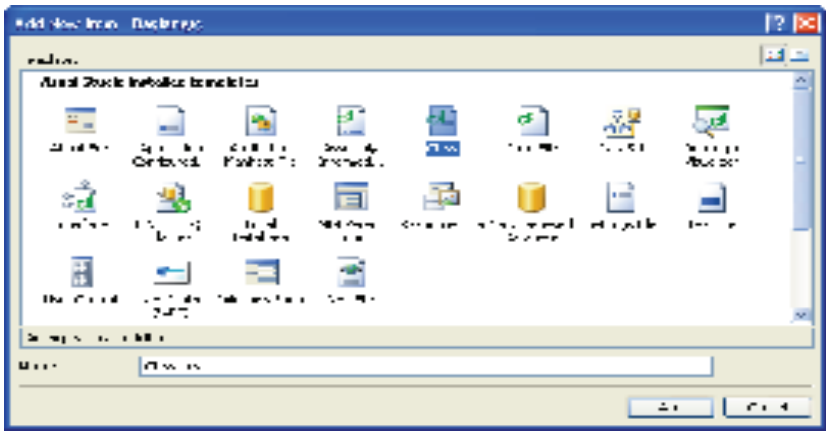
Blok Girintileri

Yeni bir deneme daha yapalım. Programın yazılış biçemindeki blok girintilerini yok edelim:

```
using System;

namespace Başlangıç{
class Program    {
static void Main(string[] args)
        {
            Console.WriteLine("Merhaba C# ");
}
}
}
```

Bu aşamada `Ctrl+E+D` tuşlarına basınız. Programda blok girintilerinin yeniden oluştuğunu göreceksiniz. `Visual Studio` ile çalışırken, programınızda blok girintileri kendiliğinden oluşur; onları yapmak için zaman ve emek harcamanıza gerek yoktur. Ancak, programda söz dizimi hatası varsa, hata giderilene kadar otomatik biçemleme yapılmaz.

Şimdi, Visual Studio ortamını biraz daha yakından tanıyalım. Ekranın sağındaki Solution Explorer penceresinde Başlangıç aduzayını sağ tıklayalım. Açılır pencerelerden Add -> Class sekmesine tıklayalım.



Önümüze Add New Item adlı şablonlar penceresi gelir. Bu pencerede C# ile yapılabilecek işlerle ilgili şablonlar yer alır. Bu şablonlar içinden Class ikonunu seçelim. Pencerenin altında Name alanında Class1.cs öndeğeri (default vakue) yer alır. Bu öneriyi kabul edebiliriz. Ama yapacağımız işi çağrıştıran bir ad vermek daha uygun olur. Class1.cs yerine Toplama.cs yazalım. Önümüze editör penceresi gelecektir. Editörle aşağıdaki programı yazalım.

Toplama.cs

```
using System;
namespace Başlangıç
{
    class Toplama
    {
        static int x = 3;
        static double y = 14.5;

        static double Topla()
        {
            short z = 5;
            return x + y + z;
        }
    }
}
```

Blok girintileri düzgün değilse Ctrl+E+D tuşlarına basarak, blokları düzgünleştirebiliriz. Yazdığımız her programı satır satır çözümlemeyi; yani her deyimin ne iş yaptığını anlamayı alışkanlık edinmeliyiz. Öğrenme sürecinde bu iyi bir alışkanlık olacaktır. Bu programı çözümlersek şunları göreceğiz. Başlangıç aduzayına ait Toplama adlı bir sınıf ile bu sınıfın içinde Topla() adlı bir metot tanımlanıyor. Sınıfın `x` ve `y` adlı iki değişkeni var. Sınıf içinde tanımlanan değişkenler ile metotlara `sınıfın öğeleri (class members)` denilir. Topla sınıfının üç öğesi var. Değişkenlere, çoğunlukla, veri alanı ya da kısaca alan (field) denilir. Bu kitapta değişken, veri alanı ve alan sözcüklerini eş anlamlı kullanacağız.

```
static int x = 3;
```

bildirimi `int` veri tipinden `x` adlı bir değişken tanımlandığını, bu değişkene ilk değer olarak `3` atandığını, değişkenin `static` olduğunu bildirir. `static` nitelemesi, daha önce de söylediğimiz gibi, Toplama

sınıfının bir nesnesini yaratmadan `x` değişkenine erişilebileceğini belirtir. `static` nitelemesini ileride daha ayrıntılı inceleyeceğiz. *Değişkene erişmek* demek, istendiğinde ona değer atamak, istendiğinde atanmış değeri okuyup işlemlerde kullanabilme yeteneğine sahip olmak demektir.

```
static double y = 14.5;
```

deyimini yukarıdaki gibi çözümlenebilir.

```
public static double Topla()
```

deyimi değer kümesi `double` olan `Topla()` metodunun bildirimidir. Bu bildirimde metodun adı, değer kümesi ve varsa parametreleri yazılır. Bunlar bir metodu kesinkes belirtirler. O nedenle

```
double Topla()
```

ifadesine metodun imzası denilir. Metotlar imzalarıyla birbirlerinden ayırt edilirler. Adları, değer kümeleri veya parametrelerinden enaz birisi farklı ise iki metot farklı olur. Örneğin, adları ve değer kümeleri ve hatta işlevleri aynı olan iki metot parametreleri farklı olduğu için birbirlerinden ayırt edilebilirler.

Metot imzasının önünde

```
public static
```

nitelemelerini görüyoruz. Bunlardan `static` nitelemesi önce söylediğimiz işleve sahiptir. `public` nitelemesi ise, metoda dışarıdan erişimi; yani onun başka sınıflardan çağrılmasına izin veren bir erişim belirtecidir. *Erişim Belirteçleri* 'ni ileride ayrıntılı göreceğiz.

Son olarak `Topla()` metodunun gövdesine bakalım. `short` tipinden `z` adlı bir değişken bildirilmiş ve ona `5` değeri atanmıştır. Bu şekilde metodun gövdesinde tanımlanan değişkenlere metodun `yerel (local)` değişkeni ya da iç değişkeni denir. Metodun alacağı değer

```
return x + y + z;
```

deyimi ile belirtilmiştir. Bunun anlamı apaçıktır, fonksiyonun değeri `x, y ,z` değişkenlerinin değerlerinin toplamıdır. Metodun değeri daima `return` anahtar sözcüğü ile belirtilir. `return` ifadesinden çıkan değer metodun *değer kümesine* ait olmalıdır. Bu örnek için söylersek, `x+y+z` toplamı, metodun değer kümesi olan `double` veri tipinden olmalıdır. Aksi halde derleme hatası doğar.

Artık `Toplama.cs` programımız tamamdır. `File -> Save Toplama.cs` sekmesini tıklayarak ya da `CTRL+S` tuşlarına basarak dosyamızı kaydedelim. Ekranın sağ yanındaki `Solution Explorer` penceresindeki `Başlangıç` aduzayı altında `Toplama.cs` programını görüyor olmalıyız.

Şimdi aynı yolları izleyerek aşağıdaki iki programı yazalım ve kaydedelim.

Çıkarma.cs

```
using System;

namespace Başlangıç
{
    class Çıkarma
    {
        static double x = 17;
        static double y = 8.5;

        public static double Çıkar()
        {
            return x - y;
```

```
        }

    }
}
```

Ana.cs

```
using System;

namespace Başlangıç
{
    class ana
    {
        static void Main(string[] args)
        {
            double y = Toplama.Topla();
            double x = Çıkarma.Çıkar();
        }
    }
}
```

`Ana.cs` programını çözümleyelim. Ana sınıfında

```
static void Main(string[] args)
```

bildirimi ile `Main()` metodu tanımlanmıştır. `Main()` metodu genellikle bu biçemde bildirilir. `Main()` metodunun `String[]` tipinden `args` adlı bir array parametresi var. Bu parametreyi çoğunlukla kullanmayacağız. Kullanmayacağımız zaman parametreyi yazmasak da olur. Ama yazmanın sakıncası yoktur. `Main()` metodu bir programı çalıştıran asıl metot olduğu için, ona ayrı bir özen gösterelim ve yukarıda gösterildiği gibi parametresini yazmayı alışkanlık edinmeye çalışalım. Main() 'in gövdesindeki deyimleri, sırasıyla, nceleyelim.

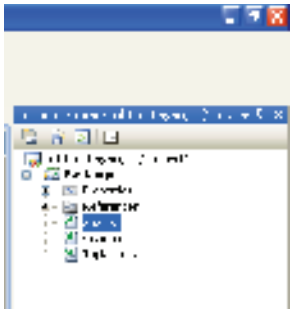
```
Toplama.Topla();
```

deyimi `Toplama` sınıfı içindeki `Topla()` metodunu çağırıyor. Bu eyleme *metot çağırma* veya *mesaj* iletme diyoruz. Bu mesajla, bir sınıftan başka bir sınıfa ait metodu çağırıp işlevini yapmasını istiyoruz. Bu mesajlaşma eylemi, nesne programlamanın (object oriented programming) üstün bir niteliğidir.

İkinci deyim olan

```
Çıkarma.Çıkar();
```

deyiminin açıklaması da benzerdir. Şimdi sağ taraftaki Solution Explorer penceresinde Başlangıç aduzayına ait olarak yazdığımız `Ana.cs, Çıkarma.cs` ve `Toplama.cs` programlarını görüyoruz.



Bunlardan herhangi birisine çift tıklarsak, editör penceresine onun kaynak programı gelir. Önümüze gelen programda, istediğimiz değişikliği yapabiliriz.

Program seçme işini editör penceresinden daha kolay yapabiliriz. Bencerenin üstündeki programlar satırında `Ana.cs, Çıkarma.cs` ve `Toplama.cs` programlarını görebiliriz. Onlardan birisine tıklarsak, o program editörde açılır. Her üç programı tam yazdığımıza emin olunca, `File -> Save All` sekmesine tıklayıp her üçünü birden kaydedelim. Hepsini birden kaydetmenin ikinci bir yolu CTRL+Shift+S tuşlarına basmaktır. Ama en kolayı, ekranın üstten üçüncü satırında yer alan sıralanmış disketler simgesine tıklamaktır. Tek disket simgesine tıklanınca, o anda editörde etkin olan dosya kaydedilir. Disketler grubuna tıklayınca, bütün dosyalar birden kaydedilir.

Programlarımızı koşturmadan önce, Visual Studio'nun birkaç aracını daha görelim. Örneğin, Ana.cs yi tıklayıp açalım.

Imleci herhangi bir stırın herhangi bir yerine koyalım. Sonra üstten üçüncü satır içindeki 5-inci ve 6-ıncı girintileme simgelerine [girinti artırıp azaltan (increase/decrease indent)] tıklayarak, girintilerin nasıl azalıp çoğaltıldığına bakalım. Eğer, Visual Studio'nun otomatik yaptığı blok girintilerinden memnun değilseniz, bu düğmeleri kullanabilirsiniz.

Tekrar imleci herhangi bir stırın herhangi bir yerine koyalım. Sonra üstten üçüncü satır içindeki 7-inci ve 8-inci simgelere [açıklama koy/kaldır (comment/uncomment)] tıklayarak, satırlara nasıl açıklama simgesi (//) konulduğuna bakalım.

Visual Studio'nun penceresine istediğiniz alt pencereleri getirebilirsiniz. Bunun için `View` açılır penceresinden istediğinizi seçiniz. Özellikle, programın koşması sırasında oluşabilecek hataları görmek için `Error List` penceresini açmalısınız.

Visual Studio'nun penceresindeki öteki simgelerin işlevlerini zamanla öğreneceksiniz. Ama çok merak edenler, `Help` sekmesinden hemen bilgi alabilirler.

Bunca sabırdan sonra, artık programlarımızı koşturmayı deneyebiliriz. Ne yapacağımızı zaten biliyoruz. `Ana.cs` programına tıklıyoruz. `Main()` metodu burada olduğu için, program girişi bu sınıftan olacaktır. Her programda bir tek giriş yeri olabilir. Birden çok `Main()` programı olsa bile, onlardan yalnızca bir tanesi giriş yeri olmalıdır.

`Ana.cs` programını çalıştırınca, onun içindeki `Main()` metodu, sırasıyla, `Toplama` ve `Çıkarma` sınıflarındaki `Topla()` ve `Çıkar()` metotlarını çağıracaktır. Dolayısıyla, yapacağımız iş, Ana.cs programını derleyip koşturmaktan ibarettir. Bunun için

```
Debug -> Start Without Debugging
```

sekmesine tıklamak ya da CTRL+F5 tuşlarına basmak yetecektir. Bunu yapınca ekrana siyah konsolun geldiğini ve içinde Hiçbir hata iletisi olmadığını göreceğiz. Öte yandan, Visual Studio ekranının altındaki Error List penceresinde de bir uyarı görülmüyor. Bu demektir ki, programlarımız derleme hatası (compile time error) vermemiştir; söz dizimleri doğrudur. Ayrıca koşma hatası (runtime error) da yoktur.

Ama konsolda programın yaptığı toplamayı ve çıkarmayı göremedik. Bunun nedeni apaçıktır. İstediklerimiz yapıldı, ama onları konsolda göremedik, çünkü konsola bir şey yazılmasını istemedik. Şimdi bu eksikliği kolayca giderebiliriz. Ana.cs programını açıp, Main() 'in gövdesindeki iki deyimi şöyle değiştirelim:

```
            Console.WriteLine(Toplama.Topla()) ;
            Console.WriteLine(Çıkarma.Çıkar()) ;
```

Bu değişikliği yapınca Ana.cs programı şu şekli alır.

Ana.cs

```
using System;

namespace Başlangıç
{
    class ana
    {
        static void Main(string[] args)
        {
            Console.WriteLine(Toplama.Topla());
            Console.WriteLine(Çıkarma.Çıkar());
        }
    }
}
```

Şimdi bunu koşturursak, konsola

```
22,5
8,5
```

yazıldığını göreceğiz.

Visual Studio Express Editions

Visual Studio Express Editions Microsoft'un Visual Studio ve SQL Server adlı ürünlerinin bir uzantısı sayılır. Öğrenciler, yeni öğrenmeye başlayanlar ve hatta programcılıkla bir hobby olarak uğraşanların dinamik Windows Uygulamaları yapmalarını, Web siteleri kurmalarını ve veritabanı uygulamaları yapmalarını kolaylaştıran bir araç, görsel bir arayüzdür. Ücretsizdir. İnternetten indirilebilir.

Visual Studio Express Editions şu bileşenlerden oluşur.

- Visual Basic 2008 Express Edition – İlk başlayanların tercih ettiği Visual Basic dili ile program geliştirme aracı
- Visual C# 2008 Express Edition – Windows uygulamaları için mükemmel bir araç. Tamamen Nesne Yönelimli bir dildir.
- Visual C++ 2008 Express Edition – C++ dilini bilenlerin tercih ettiği bir yarış atıdır.
- Visual Web Developer 2008 Express Edition – Dinamik web uygulamaları geliştirmeye yarayan kolay kullanılır bir araç.
- SQL Server 2008 Express and SQL Server Compact Edition – Veritabanı uygulamaları için kullanımı kolay ve güçlü bir araç

Visual Studio Express Editions yukarıdaki bileşenleri .NET platformu denilen tek bir platforma taşır.

.NET Framework hakkında aşağıda yeterli bilgi verilecektir.

Sıkça Sorulan Sorular

1. C# kaynak programını yazdım, derledim ve koşturdum. Bu eylemler diğer dillerde yaptıklarımdan farklı görünmüyor. Öyleyse .NET Framework ne işe yarıyor?

   .NET Framework'un mimarisi klâsik Windows uygulamalarına çok benzer. Bir .NET uygulamasını derlediğimizde ona ait .exe ya da .dll library dosyası ile birlikte AssemblyInfo dosyası oluşur. Bu dosya Win32 header, metadata, manifest ve MSIL kodlarını içerir. Biz .exe dosyasını çalıştırınca Win32 header dosyası CLR'i (Common Language Runtime) harekete geçirir, ona uygulamanın giriş noktasını gösterir. CLR o andan sonra JIT (Just-in-Time) derleyicisini ateşler. JIT, CLR 'dan gelen kodları işletim sisteminin anladığı makina diline dönüştürür. Görüldüğü gibi, biz bir programı derleyip koştururken .NET 'in (Framework, CLR, JIT, MSIL) bileşenleri devrededir, ancak onlar perdenin arkasındadırlar, biz görmeden rollerini oynarlar.

2. `Console` sınıfına ait bir nesne yaratmadığım halde, o sınıfın `Write()`, `WriteLine()`, `Read()`, `ReadLine()` metotlarını nasıl kullanabiliyorum?

   Bu metotlar `System.Console` sınıfının statik öğeleridir. Bir sınıfa ait nesne yaratılmadan o sınıfa ait statik öğeler kullanılabilir. Bu kavram ilerideki konularda ayrıntılı olarak açıklanacaktır.

3. Bir programda birden çok sınıf varsa, derleyici, `Main()` metodunun hangi sınıfta olduğunu nasıl biliyor? Birden çok sınıfta `Main()` metodu varsa, hangisini seçiyor?

   `Main()` metodunun hangi sınıfta olduğunu derleyiciye biz bildiriyoruz. Örneğin, ProgramDeneme.cs adlı programda `Main()` metodu Uygulama adlı sınıfın içindeyse ve programı satır komutuyla derliyorsak,

   ```
   csc ProgramDeneme.cs /main:Uygulama
   ```

   komutunu veririz. Eğer Visual Studio'yu kullanıyorsak

   Solution Explorer -> Properties -> Startup

   sekmesinden giriş noktası (Main() metodunu içeren sınıf) seçilebilir.

4. Bir sınıfın içine birden çok Main() metodu konulabilir mi?

   Hayır. Bir sınıfın içinde aynı adı taşıyan birden çok öğe olamaz. [Aşkın (overloaded) kavramını ileride ele alacağız.]

5. Komut satırı ile derleme ve koşturma eylemlerini ayrı ayrı yapabiliyoruz. Visual Studio'da programı derleme ve koşturma eylemlerini ayrı ayrı yapabilir miyiz?

   Evet. Programı derlemek için

   Build -> Build Solution

sekmesine tıklarız. Aynı işi F6 tuşu ile ya da Ctrl+Shift+B tuşlarıyla da yapabiliriz. Derlenen programı koşturmak için Debug -> Start Without Debug sekmesine tıklarız. Program önceden derlenmemişse, bu komut önce derleme eylemini sonra koşturma eylemini yapar. Aynı işi Ctrl+F5 tuşlarına basarak da yapabiliriz.

6. Visual Studio Express Editions hakkında nerelerde bilgi bulabilirim?

   Microsoft'un http://www.microsoft.com/express/ adresinde en son sistematik bilgileri ve hemen her konuyla ilgili öğretici dökümanları bulabilirsiniz. Ayrıca Visual Studio Express Editions kullanıcısı olarak kaydınızı yaptırabilirsiniz. Böylece bir çok programı ücretsiz edinebilirsiniz. Bu bağlamda

   ```
   MSDN 2008 Express Edition Library
   ```

   herkes için mükemmel bir kaynaktır. Ayrıca

   http://www.microsoft.com/express/vcsharp/

   adresinden her zaman bilgi alabilirsiniz.

7. *Visual Studio Express Editions* ile ticari programlar yazabilir miyim?

   Evet, yazabilirsiniz. Bunu yaptığınız için kimseye telif veya lisans ücreti ödemek zorunda değilsiniz.

8. Visual Studio Express Editions ile Visual Studio arasındaki fark nedir?

   Visual Studio profesyonel programcılar için daha çok materyale sahiptir. Visual Studio Express Editions yeni başlayanlar içindir. Expressi öğrenenler, isterlerse Visual Studio'ya yükselebilirler (upgade). Express ile yazılan programlar, kolayca Visual Studio ortamına taşınabilir. Tabii, Visual Studio'nun ücretli olduğunu söylemeye gerek yok.

Bölüm 03

Sınıflar ve Nesneler

Sınıf Nedir?

Sınıf (class) soyut bir veri tipidir. Nesne (object) onun somutlaşan bir cismidir.

Bu tanımın kendisi de çok soyut kalıyor. O nedenle bir örnekle biraz ayrıntıya inelim. *'motorlu taşıtlar'* denilince her birimizin kafasında bir kavram (sınıf) belirir. Muhtemelen ilk aklımıza gelenler motorsiklet, otomobil, otobüs, kamyon, tren gibi karada yürüyen taşıtların oluşturduğu sınıftır. Ama biraz düşününce bunlara denizde ve havada gidenleri de ekleyebiliriz. Bütün bunların ortak bir özeliği vardır. Hepsinin akaryakıtla çalışan motorları vardır. 'Motorlu taşıtlar' demek yerine, 'motorlu kara taşıtları' dersek sınıfa giren öğeleri biraz sınırlamış oluruz. Bu sınıftakilerin iki, üç, dört ya da daha çok tekerlekleri olduğunu, karayollarında yürüme yetenekleri olduğunu, yük veya insan taşıdıklarını anlarız. Motorlu taşıtlar sınıfı, motorlu kara taşıtları sınıfının bir üst sınıfıdır. Kavramı biraz daha daraltalım ve 'otomobiller' diyelim. Bu sınıf, motorlu kara taşıtları sınıfının bir alt sınıfıdır. Artık hepimizin kafasında bu sınıf oldukça iyi şekillenir. Ama hâlâ seçenekler çoktur. Otomobillerin markaları, modelleri, renkleri, motor güçleri, vb nitelikleri farklıdır. Belli bir markayı, örneğin BMW marka otomobiller sınıfını düşünürseniz, bu sınıfa ait olan araçların özeliklerini daha iyi biliyor olacaksınız. Bu sınıfı model, renk, bulunduğu ülke vb

nitelemelerle istediğiniz kadar daraltabilirsiniz. Ama bir otomobil sınıfı somut bir otomobile (bir cisim olarak) eş değildir.

Öte yandan, diyelim, Ahmet Bey'in BMW otomobili somut bir varlıktır. Bu otomobil, BMW marka otomobiller sınıfına, otomobiller sınıfına, motorlu kara taşıtları sınıfına ve motorlu taşıtlar sınıfına ait bir nesnedir, somut bir varlıktır. Bu otomobilin motor gücü, lastik ebadı, göstergelerin yeri ve biçimi gibi nitelikleri ait olduğu sınıf tarafından belirlenir.

Görüldüğü gibi, sınıf kavramı belli bir veya birden çok ortak özellikleri olan varlıkları tanılamaya yarayan soyut bir kavramdır, bir kümedir. Nesne ise bir sınıfa ait belirgin bir öğe, bir varlıktır.

Nesne Yönelimli (object oriented – OO) Programlama kavramı, bir önceki kavram olan yapısal programlama kavramının doğal bir genişlemesidir. Algol ve benzeri dillerin etkisiyle 1970 li yıllarda ortaya çıkan yapısal programlama kavramı Pascal dilinde *record*, C dilinde *struct* diye adlandırılan soyut veri tipini yarattı. Bunu şöyle açıklamaya çalışalım. Yapısal bir dilin olanaklarını kullanmadan, örneğin, bir otomobilin markasını, modelini, rengini, lastik ebadını, motor gücünü ayrı ayrı birer değişkenle belirleyebilirsiniz. Ama, belirlemek istediğiniz bütün nitelikleri tutacak değişkenleri bir araya getiren soyut bir veri yapısı kurarsanız, o yapıyı istediğiniz her otomobilin niteliklerini belirlemek için istediğiniz kadar ve istediğiniz her yerde kullanabilirsiniz. Yapı içinde tuttuğunuz değişkenler üzerinde işlemler yapmak gerektiğinde, o işi yapacak fonksiyonları programda tanımlamanız mümkündü.

1995 yılından sonra yaygınlık kazanmaya başlayan nesne yönelimli programlama, bir adım daha ileri giderek, 1970 lerde yaratılan soyut veri yapısına değişkenler yanında, o değişkenlerle işlem yapacak fonksiyonları da ekledi. Örneğin, otomobil örneğine dönersek, onun markası, modeli, rengi gibi özeliklerini değişkenlerle belirtebiliriz. Bunun yanında, kontak anahtarını çevirince motor çalışır, gaza basınca araba hızlanır, direksiyon çevrilince otomobil döner, frene basınca durur. Bunlar otomobilin yaptığı hareketler, eylemlerdir.

Bir otomobil için marka, model, renk gibi niteliklerin her birisi otomobilin bir özeliğidir. Her özeliği bir değişkenle belirleriz. Otomobilin yürümesi, hızlanması, dönmesi, durması gibi hareketler onun eylemleridir, davranışlarıdır. Otomobilin her bir eylemini bir fonksiyonla belirleriz.

Bir varlık için, istenen özelikleri tutan değişkenleri ve eylemleri belirleyen fonksiyonları içeren soyut veri yapısı bir *sınıftır*. Burada 'varlık' somut bir şey olmak zorunda değildir. Düşünsel, soyut bir varlık da olabilir.

Bu soyut yapı (sınıf) belirlendikten sonra, Ahmet Bey'in otomobilinin niteliklerini belirlemek için, o soyut tasarımdan somut bir otomobil üretmemiz gerekir. Otomobili üretince onun modelini, rengini, döşemelerini, göstergelerini vb belirgin kılabiliriz.

Sınıf Bildirimi

Bu bölümde C# dilinde sınıf bildirimini, new operatörü ile sınıftan nesne yaratmayı, sınıfın öğelerine erişimin nasıl olduğunu öğreneceğiz.

C# dilinde sınıf bildirimi (tanımı) çok kolaydır. Örneğin,

```
class Ev
{

}
```

deyimi bir sınıf bildirir. Burada `class` anahtar sözcüktür. `Ev` sınıfın adıdır. İstediğiniz adı verebilirsiniz. { } blokuna `sınıfın gövdesi` yada (sınıf bloku) diyeceğiz.

C# derleyicisi, kaynak programdaki birden çok ardışık boşluk, tab, ve satırbaşını tek bir boşluk olarak algıladığını söylemiştik. Dolayısıyla, yukarıdaki deyimi

```
class  Ev { };
```

biçiminde de yazabiliriz. Ama kaynak programımızın kolay okunup anlgılanabilmesi için her deyimi ayrı satıra yazmaya, iç-içe blokları tab ile görünür biçime getirmeye özen göstereceğiz.

Sınıf gövdesi yukarıdaki gibi boş olabilir. Ama boş gövdeli sınıfın bir işe yaramayacağı açıktır. Onun işe yarar olabilmesi için içine sabit, değişken ve fonksiyon bildirimleri (tanımları) koyacağız. Onlara `sınıfın öğeleri` (class member) denilir.

Şimdi `Ev` sınıfımızı işe yarar hale getirmeye çalışalım. Bir ev ile ilgili bilgileri tutan değişkenleri ve o bilgilerle işlem yapan fonksiyonları `Ev` sınıfının gövdesine yerleştireceğiz. Örneğin, bir evin adresi, kapı numarası, oda sayısı, evin sahibi, emlak vergisi, fiyatı vb bilgilerden istediklerimizi sınıfın gövdesine koyabiliriz. Konunun basitliğini korumak için, başlangıçta yalnızca kapı numarasını ve sokağın adını yazalım. Her evin bir kapı numarası vardır. Bu numara bir tamsayıdır. `byte, short, int` veya `long` tipinden olabilir. `int` tipini seçelim. Sınıfımız şöyle olacaktır.

```
class Ev
{
      int kapıNo ;
      string  sokakAdı ;
}
```

Bu sınıfın içine başka öğeler koymak, `kapıNo`' yu yazmak kadar kolaydır. Ona benzer işleri ileride bolca yapacağız.

Sınıf ve Nesne

Önce yapacağımız işin gerçek yaşamda neye benzediğini bir örnekle açıklamaya çalışalım. Bir ev yapmak istiyoruz. Önce evin mimari tasarımını yaparız. O tasarım evin konumunu, odalarını, mutfağını, banyosunu, pencerelerini, nasıl ısıtılacağını, nasıl havalanacağını, suyun, elektriğin, doğal gazın nerelerden nasıl geleceğini, vb belirler. Bu tasarım, büyük bir yapı için uzmanların tasarladığı ve çizdiği karmaşık bir mimari proje olabileceği gibi, bir köylünün kendi kafasında tasarladığı basit bir klübe de olabilir. İster büyük, karmaşık bir yapı, ister bir klübe olsun, önce ortada bir tasarım vardır. İster kağıt üzerine çizilsin, ister birisinin kafasında tasarlanmış olsun, o tasarım somut bir ev, bir nesne değildir. O tasarımın içine girip salonunda oturamaz, mutfağında yemek pişiremeyiz. O işleri yapabilmemiz için, her şeyden önce, tasarımı yapılan evin inşa edilmesi gerekir.

Gerçek yaşamda, bir mimari tasarımdan bir ev inşa etmek istediğimizde, bu işi yapan bir şirkete veya ustaya başvururuz. Şirket veya usta, tasarlanan evin somut bir örneğini yapıp anahtarı bize teslim eder.

Evin mimari tasarımını, C# dilinde bir sınıfa (class) benzetebiliriz. Mimari tasarımdan inşa edilen somut bir evi de C# dilinde bir sınıftan elde edilen bir nesneye (object) benzetebiliriz. Bir sınıf bir tasarımdır. Onu somut olarak kullanamayız. Onun için sınıftan nesne(ler) kurmalıyız.

Bir mimari tasarımdan aynı tipte istediğimiz kadar ev inşa edebiliriz. Örneğin bir sitedeki veya bir apartmandaki evlerin hepsi bir tek tasarımdan üretilir. Ancak, o evlerin her birisi kendi başına somut bir varlıktır, herbiri uzayda farklı bir yer işgal eder. Evler aynı yapıya sahiptirler, ama her bir evin boyası, badanası, içindeki eşyalar, insanlar bir başka evin içindekilerden farklıdır. Bu öznitelikler, bir evi ötekinden ayırır.

Benzer olarak, bir sınıftan istediğimiz kadar nesne kurabiliriz. O nesnelerin yapıları aynıdır, ama öznitelikleri farklıdır.

C# uygulamalarının mutlaka `Main()` metodu (fonksiyon) tarafından başlatıldığını biliyoruz. O halde, sınıftan nesne kurmak istediğimizde ona başvurabiliriz. `Main()` metodu kendi başına ortalıkta duramaz, o da mutlaka öteki öğeler gibi bir sınıf içinde olmalıdır. O nedenle, `Main()` metodunu içeren bir sınıf yaratmalıyız. `Uygulama` adlı boş bir sınıf yaratalım:

```
class Uygulama
{
}
```

Sonra bu sınıfın gövdesine Main() metodunu ekleyelim.

```
static public void Main()
{
}
```

new Operatörü ile Nesne Yaratmak

`Main()` metodunun gövdesine, `Ev` sınıfından bir nesne yaratacak (kuracak) olan deyimleri koyalım.

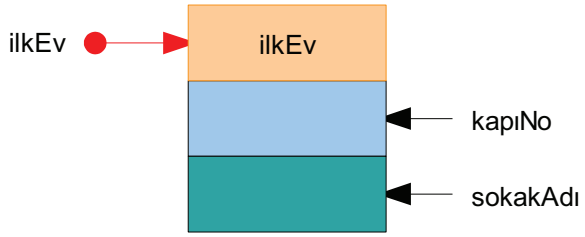
```
Ev ilkEv ;                                                    (1)
```

Bu deyim derleyiciye `Ev` sınıfına ait bir nesnenin ana bellekteki adresini gösterecek bir referans bildirimidir. Çeşitli kaynaklarda buna `pointer`, `gösterici`, `işaretçi`, `referans` gibi adlar verilir. C# dili `'reference'` sözcüğünü kullanır. Biz de ona uyarak `'referans'` veya `'işaretçi'` sözcüklerini eşanlamlı olarak kullanacağız. Bunun tek ve önemli görevi `Ev` sınıfına ait bir nesnenin bellekteki adresini işaret etmektir.

(1) bildirimi yapıldığında, henüz işaret edilecek somut bir Ev yoktur. Dolayısıyla, o aşamada, `ilkEv` işaretçisi bellekte hiç bir yeri işaret etmez. Hiç bir yeri yeri işaret etmediğini belirtmek için, işaretçiye `null` işaret ediyor deriz.

Şimdi işaretçimizin işaret edeceği somut bir Ev inşa etmeliyiz. Somut bir Ev'i inşa etmek demek, ana bellekte Ev'in bütün öğelerinin sığacağı bir yeri tahsis etmek demektir. Bu yere Ev'in bir nesnesi (object, instance) denir. Yapılan bu işe de sınıfa ait bir *nesne yaratmak (instantiate)* denilir. Bellekte sınıfa ait bir nesneyi yaratırken, bir inşaat ustası kadar uğraşmayacağız. Şu basit deyimi yazmamız yetecektir.

```
IlkEv = new Ev() ;                                            (2)
```



Bunu yapınca, ana bellekte yukarıdaki şeklin gösterdiği olaylar olur. Elbette ana bellekte böyle geometrik şekiller oluşmuyor. Ama şekiller ve resimler soyut kavramları algılamamızı kolaylaştırır. O nedenle, şeklimize bakmaya devam edelim. Ev sınıfının iki öğesini tanımlamıştık: kapıNo ve sokakAdı. Bellekte yaratılan nesneye bakarsak, orada kapıNo ve sokakAdı için ayrı ayrı yerler açılmış olduğunu görüyoruz. Bir sınıfa ait nesne (object) yaratmak demek, ana bellekte sınıfa ait static olmayan bütün öğelere birer yer ayırmak demektir. Ayrılan bu yerlere başka değerler yazılamaz; o nesne bellekte durduğu sürece, yalnızca ait oldukları öğelerin değerleri girebilir. Bunu, bellekte yer kiralama gibi düşünebiliriz. Kiralanan yerde ancak kiracı oturabilir. Henüz yaratılan nesnenin öğelerine değer atanmamıştır; yani kiracı henüz taşınmamıştır. Bize göre kapıNo ve sokakAdı değişkenleri için ayrılan hücreler boştur. Ama, C# bellekte yaratılan her değişkene kendiliğinden bir öndeğer (default value) atar. Öndeğer veri tipine göre değişir. Aşağıdaki tablo başlıca veri tiplerine atanan öndeğerleri göstermektedir.

Veri Tipi	Değer
byte, short, Int, long	0
float, double	0,0
bool	False

char	'\0' (null karakter)
string	"" (boş string)
nesne (object)	null

`KapıNo` değişkeni `int` tipi olduğu için onun öndeğeri 0 dır. `sokakAdı` ise `string` tipi olduğundan öndeğeri "" (boş string) dir. Olayın basitliğini korumak için, öndeğerleri şekle yazmıyoruz. Zaten, birazdan onların gerçek değerlerini atayacağız.

İstersek (1) ve (2) deyimlerini birleştirip, iki işi tek adımda yapabiliriz.

```
Ev ilkEv = new Ev();                                    (3)
```

Sonuncu deyim, ilk iki deyimin yaptıklarına denk iş yapar.

Bu deyime `nesne yaratıcı (instantiate)` diyoruz. Sözdizimine bakınca ne yaptığını anlamak mümkündür. Bu deyimde yer alan sözcüklerin işlevlerini soldan sağa doğru şöyle açıklayabiliriz:

Ev	Yukarıda tasarladığımız Ev sınıfıdır;
ilkEv	Tasarımdan üretilecek olan somut evin yerini işaret eder. Bu nedenle, ilkEv'i işaret ettiği evin adıymış gibi de düşünebiliriz. Bundan böyle işaretçi (referans) adı ile işaret ettiği nesneyi aynı adla anacağız. Söylemlerimizde, kastedilen şeyin referans mı, nesne mi olduğu belli olacaktır. Ancak çok gerektiğinde, referans oluşuna ya da nesne oluşuna vurgu yapacağız.
=	Atama operatörü
new	Sınıftan nesne yaratan operatör (nesne yaratıcı)
Ev()	Yaratılacak nesnenin tasarımı

Bir sınıftan bir nesne yaratan genel sözdizimi (syntax) şöyledir:

```
sınıf_adı   nesne_adı = new       sınıf_adı();
```

Şimdiye kadar söylediklerimizi yapmak için aşağıdaki Uygulama sınıfını yazmak yetecektir. Tabii, Ev sınıfını daha önce yazmıştık.

```
class Uygulama
{
    static public void Main(string[] args)
    {
        Ev ilkEv = new Ev();
    }
}
```

Şu ana kadar iki sınıf tanımladık. `Ev` adlı sınıf bir evin kapı numarası ile bulunduğu sokağı tutacak iki değişkene sahiptir. Ama `Ev` sınıfı bir tasarımdır, kendi başına bir iş yapamaz.

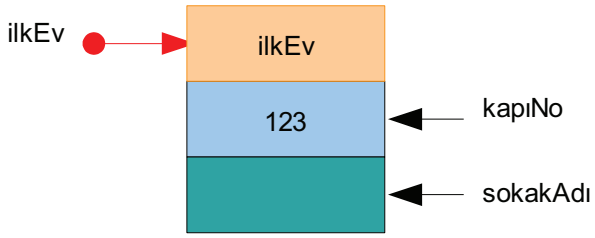
`Uygulama` adlı sınıfta `Main()` metodu tanımlıdır ve bu metot `Ev` sınıfının bir nesnesini yaratacak olan *nesne yaratıcıyı* çalıştırmaktadır. Nesne yaratıldıktan sonra, onun öğeleriyle ilgili işleri yapmaya başlayabiliriz.

Nesnenin Öğelerine Erişim

İlk işimiz nesnenin öğelerine (değişken) birer değer atamak olmalıdır. ilkEv nesnesi içindeki kapıNo ve sokakAdı bileşenleri birer değişkendir. Dolayısıyla, değişkenlerle yapılan her iş ve işlem, onlara da uygulanabilecektir. Birincisi `int` tipinden, ikincisi `string` tipinden olduğuna göre, onlara tiplerine uygun birer değer atayabiliriz:

```
ilkEv.kapıNo = 123;
```

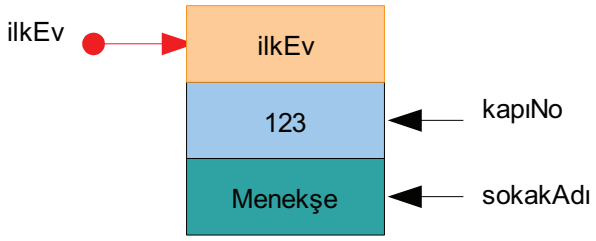
Bu atamadan sonra `ilkEv` nesnesinin durumu aşağıdaki şekildeki gibidir.



Sonra şu atamayı yapalım.

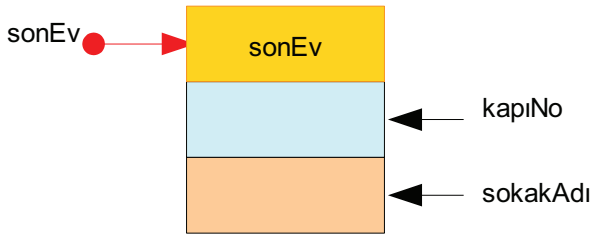
```
ilkEv.sokakAdı = "Menekşe";
```

Bu atamadan sonra `ilkEv` nesnesinin durumu aşağıdaki şekildeki gibidir.

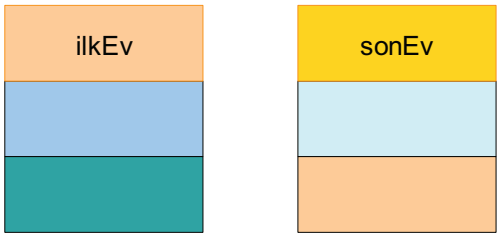


Nesne içindeki değişkenlere yapılan atama deyimlerinde, değişken adlarının önüne nesne adını yazıyoruz. Bunun nedeni açıktır. Diyelim ki,

```
Ev sonEv = new Ev();
```



nesne yaratıcısını kullanarak başka bir nesne yarattık. O zaman, bellekte iki nesne ve her birinde `kapıNo` ve `sokakAdı` bileşenleri ayrı ayrı yer alacaktır.



O durumda, yalnızca,

```
kapıNo = 123;
sokakAdı = "Menekşe";
```

atama deyimlerini yazarsak, derleyici, bu değerleri hangi ev için atayacağını bilemez, hata iletisi verir. Bu nedenle, önce nesneyi, sonra değişkeni yazıyoruz. Tabii, aralarına (.) koymayı unutmuyoruz.

```
ilkEv.kapıNo
```

ifadesi, `ilkEv` nesnesi içindeki `kapıNo` değişkeninin adıdır. Benzer şekilde,

```
sonEv.kapıNo
```

ifadesi `sonEv` nesnesi içindeki `kapıNo` değişkeninin adıdır. Görüldüğü gibi, farklı nesneler içinde aynı adı taşıyan değişkenlere farklı değerler atanabiliyor. Değişken önüne konulan nesne adları, aynı adlı değişkenleri birbirlerinden ayırır. Zaten, bir sınıftan istediğimiz kadar farklı nesne yaratabiliyor olmamızın nedeni budur. Her nesne kendi başına bir varlıktır.

Şimdiye kadar söylediklerimizi bir program haline getirelim.

Ev.cs

```
using System;

class Ev
{
    public int kapıNo;
    public string sokakAdı;
}

class Uygulama
{
    static public void Main()
    {
        Ev ilkEv = new Ev();
        ilkEv.kapıNo = 123;
        ilkEv.sokakAdı = "Menekşe";
        Console.WriteLine("ADRES: " + ilkEv.sokakAdı + " Sokak, No: " +
ilkEv.kapıNo);
    }
}
```

Çıktı

```
ADRES: Menekşe Sokak, No: 123
```

Bu programı satır satır inceleyelim. `Ev` ve `Uygulama` ayrı ayrı iki sınıftır.

Kapsülleme (encapsulation)

Ev sınıfında dikkatimizi çekmesi gereken şey sınıfın değişkenlerine `public` sıfatının verilmiş olmasıdır.

```
class Ev
{
    public int kapıNo;
    public string sokakAdı;
}
```

Değişken bildirimindeki public nitelemesini kaldırıp programı derlemeyi deneyiniz. Derleyiciden şu hata iletisini alacaksınız.

```
Error  1    'Ev.kapıNo' is inaccessible due to its protection level      ...
```

Bu ileti bize, `Ev` sınıfının `kapıNo` adlı değişkenine erişilemediğini söylüyor. Bu olgu, nesne yönelimli programlamada adına *kapsülleme* (*encapsulation)* denen iyi bir güvenlik yöntemidir. Encapsulation kavramını basite indirerek şöyle açıklayabiliriz. Bir sınıf içindeki öğelere dışarıdan erişilememesi için sınıfın dış dünyaya kapatılması demektir. Bunu ileride daha iyi anlayacağız. Bazı durumlarda sınıfın öğelerine erişimin olması istenir. Bunu yapmak için, yukarıda kullandığımız `public` nitelemesi yeterlidir. Bunun dışında başka nitelemeler de vardır. Onları *erişim belirteçleri* bölümünde açıklayacağız.

Uygulama sınıfındaki `Main()` metodu

```
Ev ilkEv = new Ev();
```

deyimi ile `Ev` sınıfından bir nesne yaratmıştır. Başka bir deyişle, ana bellekte `kapıNo` ve `sokakAdı` değişkenleri için birer yer ayırmıştır. Bellekteki bu adreslere erişmek için `ilkEv.kapıNo` ve `ilkEv.sokakAdı` işaretçileri (referans, pointer) kullanılmaktadır. Nesne yaratıcıdan sonra gelen iki deyim, yukarıda açıklanan iki atamayı yapmaktadır. Son satır ise, atanan bu değerleri konsola yazdırır.

Aşkın Operatör (overloaded operator)

Çıktıyı konsola yazdıran `Console.WriteLine()` metoduna parametre olarak

```
("ADRES: " + ilkEv.sokakAdı + " Sokak, No: " + ilkEv.kapıNo)
```

ifadesini yazdık. Bunlardan ikisi string, ikisi değişkendir; değişken değerleri de konsola string olarak yazılacaktır. Dört stringi ard arda birleştirip yazmak için (+) operatörünü kullanıyoruz. Elbette bu eylemde (+) operatörü sayılardaki işlevinden farklı bir işlev üstlenmiş, metinleri uc uca birleştirmiştir. Buna benzer olarak, bir operatöre farklı veri tipleri üzerinde farklı işlevler yaptırılabilir. Örneğin, (+) operatörüne sayısal veri tiplerinde bilinen toplama işlemini, string veri tiplerinde metin birleştirme işlemini, tarih veri tipleri üzerinde tarih toplama işlemini yaptırırız. Farklı veri tipleri üzerinde farklı işlevler yüklenmiş operatöre *aşkın operator (overloaded operator)* denilir.

Yer Tutucu Operatör

Çıktıyı konsola yazdırırken, çoğu kez, yukarıdaki gibi (+) aşkın operatörünü kullanırız. Ama çok işlevsel bir operatörümüz daha var: Yer Tutucu operatör {}. Daha önce bu parantezleri programdaki blokları belirlemek için kullandık. Bunun başka işlevleri olduğunu göreceğiz. Onlardan önemli birisi, konsol çıktılarında değişkene yer tutmaktır. Bunu bir örnekle açıklayalım. Yukarıdaki örnekte çıktıyı

```
Console.WriteLine("ADRES: " + ilkEv.sokakAdı + " Sokak, No: " +
ilkEv.kapıNo);
```

deyimi ile yazdırdık. Aynı işi

```
Console.WriteLine("ADRES: {0} Sokak, No: {1} " , ilkEv.sokakAdı ,
ilkEv.kapıNo  );
```

deyimi de yapar. "ADRES: {0} Sokak, No: {1} " stringi içerisine iki tane yer tutucu operatör konulmuştur: {0} ve {1}. Stringden sonra virgülle ayrılmış iki değişken adı vardır. C# onlardan ilkini; yani ilkEv.sokakAdı değişkenini 0-ıncı değişken olarak sayar ve onun değerini {0} yer tutucusunun bulunduğu yere koyar. Benzer olarak, ilkEv.kapıNo değişkenini 1-inci değişken olarak sayar ve onun değerini {1} yer tutucusunun bulunduğu yere koyar. Genel olarak {n} yer tutucusu, saymayı sıfırdan başlatırsak, n-inci değişkene yer ayırır. Yer tutucu operatörle, çıktıyı nasıl biçemleyebileceğimizi ileride göreceğiz. Şimdilik, aynı çıktıyı,

```
Console.WriteLine("ADRES: {1} Sokak, No: {0} " , ilkEv.kapıNo   ,
ilkEv.sokakAdı);
```

deyimiyle de yapabileceğimizi apaçık görebiliriz.

İpucu

Yukarıdaki programda 123 ve "Menekşe" değerleri ilkEv adlı nesne içinde atanmış değerlerdir. Ev sınıfından başka nesneler yaratılabilir. Yaratılacak başka nesnelerde, bu atamalar geçerli değildir. Her nesnenin öğelerine ayrı ayrı atamalar yapılmalıdır.

Başka bir ev nesnesi yaratılırsa, onun kapı numarasının ya da sokakAdı adının farklı olması zorunlu mu? sorusu akla geliyor. Tabii gerçek hayatta, aynı sokakta kapıNo'ları eşit iki evin olması postacıyı yanıltabilir. Ama, işaretçileri (adları) farklı olduğu sürece nesneye ait değişkenlerin aynı değeri almalarına hiç bir engel yoktur. Farklı nesnelerde aynı adı taşıyan değişkenler farklı ya da aynı değeri alabilirler. Örneğin Menekşe sokakta birden çok ev olabilir. O evlerin sokakAdı adları aynı olacaktır.

Bunu görmek için, ikinciEv ve üçüncüEv adlı iki nesne daha yaratalım ve programı aşağıdaki gibi değiştirelim.

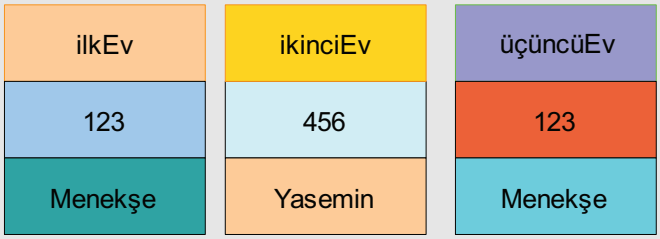
Ev02.cs

```
using System;

namespace Sınıflar
{
    class Ev
    {
        public int kapıNo;
        public string sokakAdı;
    }

    class Uygulama
    {
        static public void Main()
        {
            Ev ilkEv = new Ev();
            ilkEv.kapıNo = 123;
            ilkEv.sokakAdı = "Menekşe";

            Ev ikinciEv = new Ev();
            ikinciEv.kapıNo = 456;
            ikinciEv.sokakAdı = "Yasemin";

            Ev üçüncüEv = new Ev();
            üçüncüEv.kapıNo = 123;
            üçüncüEv.sokakAdı = "Menekşe";

            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", ilkEv.kapıNo,
ilkEv.sokakAdı);
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ",
ikinciEv.kapıNo, ikinciEv.sokakAdı);
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ",
```

```
            üçüncüEv.kapıNo, üçüncüEv.sokakAdı);
        }
    }
}
```

Çıktı

```
ADRES: Menekşe Sokak, No: 123
ADRES: Yasemin Sokak, No: 456
ADRES: Menekşe Sokak, No: 123
```



Buradan şu kuralı çıkarabiliriz.

Kural

Bir sınıftan *nesne yaratıcı* ile istediğimiz kadar nesne yaratabiliriz. Her nesneye bellekte ayrı yerler ayrılır. Dolayısıyla, birisinin değişkenlerine atanan değerler, öteki nesneleri etkilemez.

Ama, istenirse bir nesnenin değerleri başkasına aktarılabilir. Örneğin, yukarıdaki programda, konsola yazdıran iki satırdan önce

```
            Ev yazlıkEv = new Ev();
            yazlıkEv = ilkEv;
            ilkEv = ikinciEv;
            ikinciEv = yazlıkEv;
```

deyimlerini ekleyelim. Çıktıda `ilkEv` ile `ikinciEv`'in değişkenlerine verilen değerlerin birbirleriyle yer değiştirdiğini görebiliriz.

```
ADRES: Yasemin Sokak, No: 456
ADRES: Menekşe Sokak, No: 123
ADRES: Menekşe Sokak, No: 123
```

Yukarıdaki takas işlemine biraz yakından bakalım. `yazlıkEv = ilkEv` atamasında olduğu gibi, bir nesne aynı sınıftan başka bir nesneye değer olarak atanabildi.

İpucu

Bir nesne aynı sınıftan başka bir nesneye değer olarak atanabilir.

Genkurucu (default constructor)

Şimdi şunu deneyelim. Nesne yaratıcısını etkisiz kılalım; yani bir nesne yaratılmasın. Onun yerine `Ev` sınıfının değişkenlerini `public static` sıfatlarıyla niteleyelim.

GenKurucu01.cs

```
using System;

namespace Sınıflar
```

```
{
    class Uygulama
    {
        class Ev
        {
            public static int kapıNo;
            public static string sokakAdı;
        }

        static public void Main()
        {
            Ev.kapıNo = 123;
            Ev.sokakAdı = "Menekşe";
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", Ev.kapıNo,
Ev.sokakAdı);
        }
    }
}
```

Çıktı

ADRES: Menekşe Sokak, No: 123

Bu programın nasıl çalıştığını açıklamak için kurucu (constructor) kavramına biraz daha eğilmemiz gerekiyor. Önceki programlarda `Ev` sınıfının bir kaç nesnesini yarattık, o nesnelerin değişkenlerine değerler atadık. Ama yukarıdaki programda apaçık bir nesne yaratmadık. Buna karşın,

```
Ev.kapıNo = 123;
Ev.sokakAdı = "Menekşe";
```

atamalarını yaptık. Oysa, daha önce söylediklerimize göre `Ev` sınıfı bir tasarımdır; ona bir şey yaptırılamazdı; ancak o sınıftan yaratılacak nesnelere bir iş yaptırılabilirdi. O zaman, nesne yaratılmadan yapılan yukarıdaki atamaları nasıl açıklayacağız?

Yukarıdaki sorunun basit bir yanıtı var. Daha önce öğrendiğimiz kural bozulmadı. Yalnızca bizim apaçık istemediğimiz bir işi derleyici kendiliğinden yaptı. Ev sınıfı için kendiliğinden bir nesne yarattı. O nesneye sınıfın adını verdi. Böylece, `Ev.kapıNo = 123;` deyiminin başındaki `Ev` adı sınıfın değil, bizden habersiz yaratılan ve aynı adı taşıyan nesnenin adıdır. Bunu kendiliğinden yapan nesne kurucuya `genkurucu (default constructor)` denilir.

İpucu

Program bir sınıfa ait hiç bir nesne yaratmıyorsa, genkurucu kendiliğinden o sınıfa ait bir nesne yaratır; yarattığı nesneye o sınıfın adını verir.

Genkurucu birden çok nesne yaratmaz.

Nesne yaratıcı ile bir nesne yaratılıyorsa, genkurucu nesne yaratmaz.

Sınıf değişkenlerine verilen değerleri istediğimizde değiştirebiliriz. Aşağıdaki örnek bunu göstermektedir.

GenKurucu02.cs

```
using System;

namespace Sınıflar
{
    class Uygulama
    {
```

```
        class Ev
        {
            public static int kapıNo = 444;
            public static string sokakAdı = "Kardelen";
        }

        static public void Main()
        {
            Console.WriteLine("ADRES: {1} Sokak, {0} " , Ev.kapıNo,
Ev.sokakAdı);
            Ev.kapıNo = 123;
            Ev.sokakAdı = "Menekşe";
            Console.WriteLine("ADRES: {1} Sokak, {0} ", Ev.kapıNo,
Ev.sokakAdı);
        }
    }
}
```

Çıktı

ADRES: Kardelen Sokak, 444

ADRES: Menekşe Sokak, 123

İpucu

Bir sınıfın static öğelerine nesne işaretçisi ile erişilemez.

Aşağıdaki programda Ev sınıfının `static` nitelemeli `sokakAdı` değişkenini, `ilkEv` işaretçisi göremiyor; derleme hatası doğuyor.

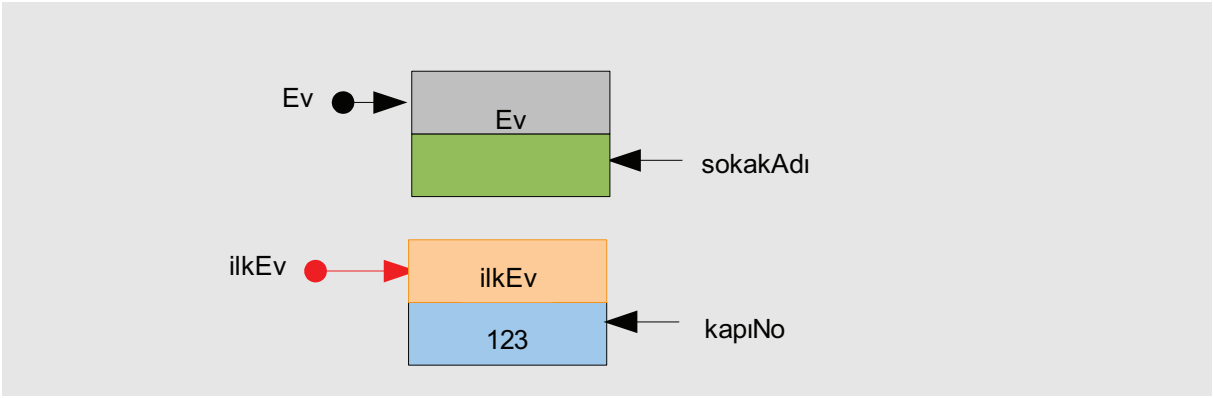
GenKurucu03.cs

```
using System;

namespace Sınıflar
{
    class Uygulama
    {
        static public void Main()
        {
            Ev ilkEv = new Ev();
            ilkEv.kapıNo = 123;
            ilkEv.sokakAdı = "Menekşe";
        }
    }

    class Ev
    {
        public int kapıNo;
        public static string sokakAdı;
    }
}
```

Error 1 Member 'Sınıflar.Uygulama.Ev.sokakAdı' cannot be accessed with an instance reference; qualify it with a type name instead ...

`Main()` metodundaki kodların yazılış sırası ile hata iletisini karşılaştırırsak şunu görebiliriz: `Ev` sınıfının `ilkEv` adlı bir nesnesi yaratılmıştır. `ilkEv.kapıNo = 123` ataması ile `ilkEv` nesnesinin `kapıNo`

adlı bileşenine `123` değeri atanmıştır. Ama `ilkEv.sokakAdı = "Menekşe"` atamasına gelindiğinde, derleyici hata vermektedir. Çünkü, Ev sınıfının `static` nitelemeli `sokakAdı` adlı öğesi bir nesne içine gitmez. Ona ana bellekte `ilkEv` nesnesi dışında bir yer ayrılır. Dolayısıyla nesne içinde `ilkEv.sokakAdı` adlı bir değişken yoktur. Bunu aşağıdaki gibi bir şekille temsil edebiliriz.



`Ev` sınıfını `Uygulama` sınıfının içine alsak bile bu durum değişmez. Böyle olduğunu görmek için, programımızı aşağıdaki gibi değiştirelim. Derleyicimizin aynı hata iletisini verdiğini göreceğiz.

GenKurucu04.cs

```
using System;

namespace Sınıflar
{
    class Uygulama
    {
        static public void Main()
        {
            Ev ilkEv = new Ev();
            ilkEv.kapıNo = 123;
            ilkEv.sokakAdı = "Menekşe";
        }

        class Ev
        {
            public int kapıNo;
            public static string sokakAdı;
        }
    }

}
```

```
Error  1  Member 'Sınıflar.Uygulama.Ev.sokakAdı' cannot be accessed with an instance reference;
```

Bir sınıfın bazı öğeleri static, bazıları değilse, *static öğelere* sınıf işaretçisi ile, *static olmayan öğelere* de nesne işaretçisi ile erişilebilir.

GenKurucu05.cs

```
using System;

namespace Metotlar
{
```

```
    class Uygulama
    {
        static public void Main()
        {
            Ev.kapıNo = 777;
            Ev eskiEv = new Ev();
            eskiEv.sokakAdı = "Yasemin";
            Console.WriteLine("ADRES: {1} Sokak, {0} ", Ev.kapıNo,
eskiEv.sokakAdı);
        }
    }

    class Ev
    {
        public static int kapıNo;
        public string sokakAdı;
    }
}
```

Çıktı

```
ADRES: Yasemin Sokak, 777
```

this *anahtarı*

Her zaman söylediğimizi anımsatarak başlayalım. C# dilinde her şey sınıflar içinde tanımlanır. Dolayısıyla, bazı dillerde önemli rol oynayan *global değişkenler* ve *global fonksiyonlar* C# dilinde yoktur. Onun yerine *sınıf öğeleri* (class member) dediğimiz değişkenler ve metotlar vardır. Sınıfın bir öğesine 'global' rol oynatmak istediğimizde, onu *public* erişim belirteci ile niteleriz; yani public nitelemeli sınıf öğelerine başka sınıflardan erişilebilir. Metotların içinde tanımlanan değişkenler o metodun yerel değişkenleridir. Yerel değişkenlere ancak ait olduğu metot içinden erişilir; kendi sınıfından veya başka sınıflardan erişilemez.

Bazı durumlarda, yerel değişken adları ile sınıf değişkenlerinin adları aynı olabilir. Bu durumda, metotlar kendi yerel değişkenlerine öncelik tanırlar. Bunu bir örnekle açıklayalım. Örneğin dairenin alanını bulan bir sınıfta `yarıÇap` adlı bir değişken hem sınıf öğesi olarak hem de bir metodun yerel değişkeni olarak bildirilmiş olsun. C#, bunları farklı iki değişken olarak yorumlar. Dolayısıyla, her ikisine ana bellekte ayrı ayrı yerler ayırır. Öyleyse, derleyici açısından onlar birbirlerinden farklı iki değişkendir.

Metotlar sınıf öğelerini görebildiğine göre, metot içinde `yarıÇap` değişkenine bir atama yapılsa, o değer hangisinin adresine yazılacak? Sınıf değişkeninin adresine mi, yoksa yerel değişkenin adresine mi? Benzer olarak, `yarıÇap` değeri okutulmak istense hangi adresteki değeri okuyacak? Derleyici bu konuda hiç tereddüde düşmez, aksi söylenmedikçe öncelik daima yerel değişkenlerindir. Böyle olduğunu aşağıdaki örnekten görebiliriz.

Daire.cs

```
using System;
class Daire
{
    public const double PI = 3.14159;
    public double yarıÇap;
    public double alan;

    public double AlanBul(double r)
    {
        double yarıÇap = r;
        double alan;
        alan = PI * r * r;
        return alan;
```

```
    }

    public void Yaz()
    {
        Console.WriteLine("Dairenin yarıçapı = {0} ", yarıÇap);
        Console.WriteLine("Dairenin alanı    = {0} ", alan);
    }
}

class Uygulama
{
    static void Main(string[] args)
    {
        Daire d = new Daire();
        Console.WriteLine(d.AlanBul(2.5));
        d.Yaz();
    }
}
```

Çıktı

```
19,6349375
Dairenin yarıçapı = 0
Dairenin alanı    = 0
```

Çıktıya bakarak bu programı çözümleyelim. `Daire` sınıfı içinde double tipinden public nitelemeli `yarıÇap` ve `alan` adlı iki *sınıf değişkeni* bildirimi yapılmıştır. Sınıfa ait metotlar ve başka sınıflara ait metotlar bu değişkenlere erişebilirler. Nitekim, `Daire` sınıfı içinde tanımlanan `Yaz()` metodu bu değişkenlere erişmekte ve onların değerlerini konsola yazmaktadır.

Gene `Daire` sınıfı içinde tanımlı olan `AlanBul()` metodu, çağrılırken aldığı parametreyi kullanarak dairenin alanını bulmaktadır. Bu metodun yerel değişkenleri, `Daire` sınıfının değişkenleri ile aynı adları taşımaktadır. `AlanBul()` metodu hesapladığı daire alanının değerini `alan` adlı yerel değişkeninin adresine yazar; çünkü öncelik kendi yerel değişkenlerinindir. Sonuçta sınıf değişkenlerine program içinde hiçbir değer atanmamıştır.

`Main()` metodu önce `Daire` sınıfına ait `d` adlı bir nesne yaratıyor. Sonra `AlanBul()` metodunun değerini olan `19,6349375` sayısını konsola yazdırıyor. En sonunda `Yaz()` metodunu çağırıyor. `Yaz()` metodu `AlanBul()` metodunun yerel değişkenlerine erişemez; o ancak sınıfın değişkenlerine erişiyor ve onların öndeğerleri (default value) olan 0 değerlerini konsola yazıyor.

Peki ama biz yerel değişkeni değil, sınıf değişkenini kullanmak istiyorsak ne yapmalıyız? O zaman `this` anahtar sözcüğü yardıma koşacaktır. Aşağıdaki örnekler bunun nasıl olduğunu gösteriyor.

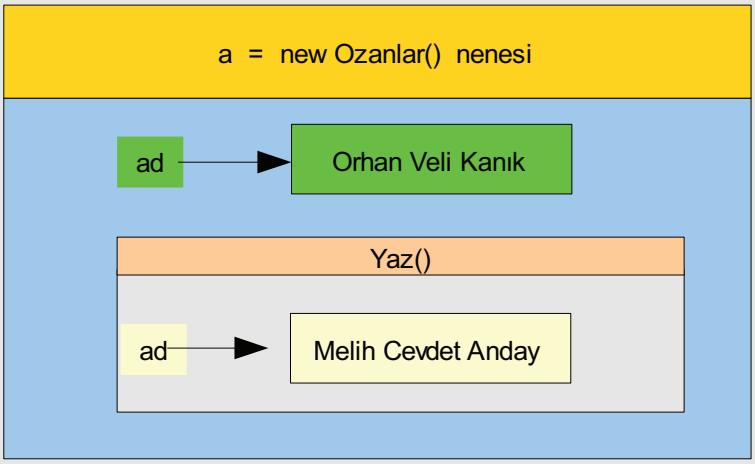
this anahtarının metot içinde kullanımı

Ozanlar.cs

```
using System;

    public class Ozanlar
    {
        public string ad  ;

        public void Yaz()
```

```
        {
            string ad = "Melih Cevdet Anday";
            this.ad   = "Orhan Veli Kanık";
            Console.WriteLine(ad) ;
            Console.WriteLine(this.ad );
        }
    }

class Uygulama
{
    public static void Main(string[] args)
    {
        Ozanlar a = new Ozanlar();
        a.Yaz();
    }
}
```

`Ozanlar` sınıfını çözümleyelim. Sınıfın içinde `ad` adlı string tipinden bir sınıf değişkeni tanımlıdır. O sınıfın bir öğesi olan `Yaz()` metodu içinde de aynı adla bir yerel değişken tanımlıdır. Derleyici her ikisine ayrı ayrı bellek adresleri ayırır.



Sınıf değişkeni ile yerel değişkene ayrı ayrı adresler tahsis edilir

```
string ad = "Melih Cevdet Anday";
```

deyimi yerel ad değişkenine "Melih Cevdet Anday" değerini atar.

```
this.ad   = "Orhan Veli Kanık";
```

deyimi ise sınıf öğesi olan ad değişkenine "Orhan Veli Kanık" değerini atar.

```
Console.WriteLine(ad) ;
```

deyimi yerel değişken değerini konsola yazar.

```
Console.WriteLine(this.ad );
```

deyimi ise sınıf değişkeninin değerini konsola yazar.

this Anahtarının Kurucu İçinde Kullanımı

Aşağıdaki programda `Ozan` sınıfı içinde tanımlanan public `Ozan()` kurucusundan sınıf değişkenine nasıl erişildiğini göstermektedir.

Ozan.cs

```
using System;
```

```
public class Ozan
{
    public string ad;

    public Ozan() //kurucu
    {
        this.ad = "Orhan Veli Kanık";
    }
}

class Uygulama
{
    public static void Main(string[] args)
    {
        Ozan obj = new Ozan();
        Console.WriteLine(obj.ad);
    }
}
```

`Ozan` sınıfı içinde tanımlanan public `Ozan()` kurucusu sınıfın ad değişkenine erişmek için `this.ad` ifadesini kullanmaktadır:

```
            this.ad    = "Orhan Veli Kanık";
```

Eğer bunlar yerine, örneğin

```
            string ad    = "Orhan Veli Kanık";
```

gibi yerel değişkenler koyarak programı

```
using System;

public class Ozan
{
    public string ad;

    public Ozan() //kurucu
    {
        string ad = "Orhan Veli Kanık";
    }
}

class Uygulama
{
    public static void Main(string[] args)
    {
        Ozan obj = new Ozan();
        Console.WriteLine(obj.ad);
    }
}
```

biçimine dönüştürürsek, program derlenir; yani sözdizimi hatası yoktur, ama çıktısı “” boş stringdir. Çünkü, `Ozan` sınıfının `ad` değişkenine değer atanmamıştır. Öndeğeri (default value) “” boş string 'dir; konsola öndeğeri gider.

Gezegen.cs

```
using System;

class Gezegen
{
    public int yarıÇap;
```

```
    public double yerÇekim;
    private string ad;
}

class Uygulama
{
    public static void Main()
    {
        Gezegen dünya = new Gezegen();
        dünya.yerÇekim = 9.81;
        dünya.yarıÇap = 6378;
        Console.WriteLine("Dünyanın yarıÇapı  = " + dünya.yarıÇap  );
        Console.WriteLine("Dünyanın yerÇekimi = " + dünya.yerÇekim );
    }
}
```

Bölüm 04

Kurucular ve Yokediciler

(Constructors and Destructors)

Kurucu Nedir?

C# dilinde iki tür kurucu vardır: dinamik (instance) kurucular, statik kurucular. Statik kurucu, öteki statik öğeler gibi davranır, ilk çağrıda sınıfa ait nesneyi yaratır. Kurucu deyince, genellikle dinamik (instance) kurucu kastedilir. Microsoft, C# kurucusunu şöyle tanımlar: *"Sınıfa ait bir nesne yaratan sınıf öğesidir."* Bu tanım güzeldir ama yeni başlayanların anlaması zor olabilir. Ama, aşağıdaki açıklamalar bu tanımı aydınlığa kavuşturacaktır.

new operatörü

Geçen bölümlerde bir sınıftan nesneler yaratmayı öğrendik, nesnelerle ilgili bazı işlemler yaptık. Bu bölümde, nesne yaratıcıların hünerlerini göreceğiz. Her zaman olduğu gibi, işin kuramsal yanını geriye bırakıp, pratik uygulamalarla kavramları açıklama yoluna gideceğiz.

Önce aşağıdaki programı derlemeyi deneyelim. Program `Ev` sınıfı içindeki `Yaz()` metodunu çağırmaktadır.

Kurucular01.cs

```
using System;
```

```
namespace Sınıflar
{
    class Uygulama
    {
        static public void Main()
        {
            Yaz();
        }
    }

    class Ev
    {
        public int kapıNo = 32;
        public string sokakAdı = "Papatya";
        public void Yaz()
        {
            Console.WriteLine("Ev {1} sokakta {0} numaralıdır.", kapıNo,
sokakAdı);
        }
    }
}
```

Program derlenirken şu hata iletisini verir.

```
Error    1    The name 'Yaz' does not exist in the current context ...
```

Buraya kadar öğrendiklerimize göre, hatanın ne olduğunu hemen anlıyoruz. `Yaz()` metodu, onu çağıran `Main()` 'in içinde bulunduğu `Uygulamalar` sınıfında değil, onun dışında olan `Ev` sınıfındadır. `Main()` metodu onu görememektedir.

`Main()` metoduna `Yaz()` metodunun `Ev` sınıfı içinde olduğunu göstermeyi deneyebiliriz. Bunun için `Main()` metodunun gövdesindeki `Yaz()` deyimi yerine

```
Ev.Yaz();
```

deyimini koyabiliriz.

Ancak, derleyicimiz, bu kez başka bir hata iletisi verecektir.

```
Error    1    An object reference is required for the non-static field, method, or property 'Sınıflar.Ev.Yaz()' ...
```

Şimdi hatayı daha iyi algılıyoruz. Derleyicimiz `Yaz()` metodunun içinde olduğu sınıfı değil, o sınıfa ait bir nesnenin işaretçisini (object reference) istemektedir. Bu isteğini de kolayca karşılayabiliriz. `yazlıkEv` adlı bir işaretçi bildirimi yapalım. Bu bildirimi yapmak için `Main()` 'in gövdesine şu deyimleri ekleyelim:

```
            Ev yazlıkEv;
            yazlıkEv.Yaz();
```

Bu kez, derleyicimiz şu hatayı iletecektir:

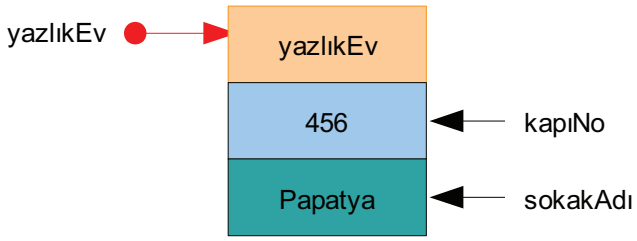
```
Error    1    Use of unassigned local variable 'yazlıkEv'    ...
```

Derleyicimizin ne dediğini artık kolayca anlıyoruz. `yazlıkEv` adlı bir işaretçi bildirimini yaptık, ama o `null` işaret ediyor; çünkü, onun işaret edeceği bir nesne yaratmadık.

Öyleyse, `new` oparatörünü kullanarak `yazlıkEv` referansının işaret edeceği bir nesne yaratalım.

```
            Ev yazlıkEv;
            yazlıkEv = new  Ev() ;
            yazlıkEv.Yaz();
```

Bu bildirimlerin etkisini aşağıdaki şekille temsil edebiliriz.



`Ev` sınıfında static olmayan `kapıNo` ve `sokakAdı` değişkenlerine verdiğimiz değerlerin, `yazlıkEv` nesnesine aynen gittiğine dikkat ediniz. Başka nesneler yaratsak, bu değerler o nesnelere de gidecektir.

İpucu

Sınıf değişkenlerine (veri alanı, field) atanan değerler, sınıfa ait her nesneye aynen gider.

Artık, derleyicimizin her istediğini yerine getirmiş oluyoruz. Aşağıdaki programımızı derleyip koşturabiliriz.

Kurucular02.cs

```
using System;

namespace Kurucular
{
    class Uygulama
    {
        static public void Main()
        {
            Ev yazlıkEv ;
            yazlıkEv = new Ev();
            yazlıkEv.Yaz();
        }
    }

    class Ev
    {
        public int kapıNo = 456;
        public string sokakAdı = "Papatya";
        public void Yaz()
        {
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }
    }
}
```

Çıktı

```
ADRES: Papatya Sokak, No: 456
```

Sınıf öğelerinin değerlerinin nesnelere aynen gittiğini söylemiştik. Program koşarken, o değerleri nesne içinde değiştirmek mümkündür; ama nesne içindeki değişiklik, sınıf tanımındaki değerleri etkilemez. Bunu aşağıdaki örnekten görebiliriz.

Kurucular03.cs

```
using System;

namespace Kurucular
{
    class Uygulama
    {
        static public void Main()
        {
            Ev  yazlıkEv = new Ev();
            yazlıkEv.Yaz();

            yazlıkEv.kapıNo = 789;
            yazlıkEv.sokakAdı = "Sümbül";
            yazlıkEv.Yaz();

            Ev kışlıkEv = new Ev() ;
            kışlıkEv.Yaz();
        }
    }

    class Ev
    {
        public int kapıNo = 456;
        public string sokakAdı = "Papatya";
        public void Yaz()
        {
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }
    }
}
```

Çıktı

```
ADRES: Papatya Sokak, No: 456
ADRES: Sümbül Sokak, No: 789
ADRES: Papatya Sokak, No: 456
```

Bu çıktıyı çözümlemek için `Main()` 'in gövdesindeki deyimlere bakalım.

```
            Ev  yazlıkEv = new Ev();
            yazlıkEv.Yaz();
```

Bu iki deyim, `yazlıkEv` adlı nesneyi yaratıyor ve onun `Ev` sınıfından aldığı değişken değerlerini konsola yazıyor:

```
ADRES: Papatya Sokak, No: 456
```

```
            yazlıkEv.kapıNo = 789;
            yazlıkEv.sokakAdı = "Sümbül";
            yazlıkEv.Yaz();
```

Bu üç deyim, `yazlıkEv` adlı nesneye `Ev` sınıfından giren değişken değerlerini değiştirip konsola yazıyor:

```
ADRES: Sümbül Sokak, No: 789
```

```
            Ev kışlıkEv = new Ev() ;
            kışlıkEv.Yaz();
```

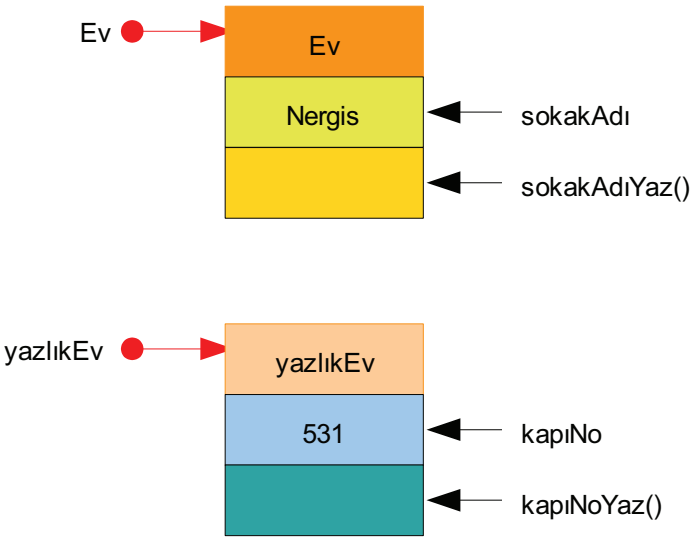
Bu son iki deyim, `kışlıkEv` adlı nesneyi yaratıyor ve onun `Ev` sınıfından aldığı değişken değerlerini konsola yazıyor.

Bu değerlerin sınıf değişkenlerinin değerleriyle aynı olduğunu görüyoruz. Bu demektir ki, ikinci grupta `yazlıkEv` nesnesi içinde yaptığımız değişiklikler sınıf değişkenlerine yansımıyor. Yansımış olsaydı, üçüncü grup çıktısı ikinci grup çıktısı ile aynı olurdu.

Static ve Dinamik Öğelere Erişim

Aşağıdaki programda `Main()` metodu `Ev` sınıfının iki metodunu çağırmaktadır. `KapıNoYaz()` metodu dinamiktir ve ona `yazlıkEv` nesnesi içinde erişilmektedir. Ama `SokakAdıYaz()` metodu static'tir; bir nesne yaratılmadan ona doğrudan erişilmektedir. Benzer şekilde, `Ev` sınıfı içinde `kapıNo` değişkeni dinamiktir, ona `yazlıkEv` nesnesi içinde erişilmektedir. Ama `sokakAdı` değişkeni static'tir; ona bir nesne yaratılmadan doğrudan erişilmektedir.

Dinamik değişken ve metotlar, sınıfa ait nesneler içinde bulunurlar. Dolayısıyla onlara nesne içinde erişilebilir. Static değişken ve metotlara, bir nesne yaratılmadan, doğrudan sınıfın adıyla erişilebilir.



Şekilde temsil edildiği gibi, `yazlıkEv` nesnesi içinde dinamik `kapıNo` değişkenine ve dinamik `KapıNoYaz()` metoduna erişilebilir. Static olan `sokakAdı` değişkenine ve `SokakAdıYaz()` metoduna nesne içinde değil, nesne dışında onlara ana bellekte ayrılan adreslerde erişilebilir. O adreslere ulaşmak için, referans olarak `Ev` kullanılır. Burada Ev, asıl sınıf değil, sözkonusu bellek adresini işaret eden referanstır.

Bu söylediklerimizi aşağıdaki programda görebilirsiniz.

Kurucular04.cs

```
using System;

namespace Kurucular
{
```

```
    class Uygulama
    {
        static public void Main()
        {
            Ev yazlıkEv = new Ev();
            yazlıkEv.KapıNoYaz();

            Ev.SokakAdıYaz();
        }
    }

    class Ev
    {
        public int kapıNo = 531;
        public static string sokakAdı = "Nergis";

        public void KapıNoYaz()
        {
            Console.WriteLine("Kapı No: {0} ", kapıNo);
        }

        public static void SokakAdıYaz()
        {
            Console.WriteLine("Sokak Adı: {0} ", sokakAdı);
        }
    }
}
```

Çıktı

```
Kapı No: 456
Sokak Adı: Papatya
```

Yukarıdaki programda static öğeleri dinamik, dinamik öğeleri static yaparak derleyicinin vereceği hata iletilerini görünüz.

Şimdi `Main()` metoduna neden `static` nitelemesi verildiğini anlıyor olmalısınız.

Kurucular (Constructors)

Önce aşağıdaki programı koşturalım, sonra çıktıyı çözümlemeye başlayalım.

Kurucular05.cs

```
using System;

namespace Kurucular
{
    class Uygulama
    {
        static void Main()
        {
            new Ev();
        }
    }

    class Ev
    {
        int kapıNo = 123;
        string sokakAdı = "Menekşe";
```

```
        public Ev()
        {
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }
    }
}
```

Çıktı

```
ADRES: Menekşe Sokak, No: 123
```

`Main()` metodu hiç bir fonksiyon çağırmadı, yalnızca `new` operatörü ile `Ev` sınıfına ait bir nesne yarattı. Nesneye bir ad verilmedi, onunla ilgili başka bir işlem yapılmadı.

`Ev` sınıfına gelince, sınıfın içinde iki değişkeni (veri alanı, field) ile görünüşü sanki bir metodu andıran

```
public Ev()
{
     Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo, sokakAdı);
}
```

bloku var. Bu blok bir metot bildirimini andırıyor, ama iki önemli farkı var:

1. Metot ait olduğu sınıfın adını taşıyor
2. Metodun bir veri tipi yok; void, int, bool gibi bir değer almayacak.

Artık kurucuyu tanımlayabiliriz.

Kurucu

Bir sınıfın kurucusu, sınıf içinde tanımlı olan, sınıfın adını alan ama değer kümesi olmayan özel bir metottur. Görevi sınıfa ait bir nesne yaratmaktır.

Yukarıdaki programa bakalım. `Main()` metodu `Ev` sınıfı içindeki `Ev()` metodunu çağırmadı; yalnızca `new` operatörü ile `Ev` sınıfına ait bir nesne yarattı. O nesneye bir ad vermedi, nesneyle ilgili başka bir iş yapmadı. Ama `Ev` sınıfı içindeki `Ev()` metodu (kurucu) kendiliğinden çalıştı ve konsola

```
ADRES: Menekşe Sokak, No: 123
```

yazdı. Bu nasıl olabildi?

Bu olgu, kurucudan beklenen önemli bir özeliktir. `Main()` metodundaki

```
new Ev() ;
```

deyimi `Ev` sınıfına ait nesneyi kurar kurmaz, onun içindeki

```
public Ev()
```

metodu *(kurucu, constructor)* kendiliğinden çalışır. O metodun (kurucu) ayrıca çağrılmasına gerek olmadığı gibi, çağrılması da mümkün değildir. Gerçekten onu çağırmayı denersek, derleyicimiz itiraz edecektir. `Main()` blokuna `new Ev()` deyimi yerine,

```
Ev aaa = new Ev();
aaa.Ev();
```

deyimlerini koyalım ve aaa adıyla bir nesne yaratalım ve o nesne içinden Ev() metodunu (kurucu) çağıralım. Derleyicinin şu hata iletisini verdiğini göreceğiz.

```
Error   1   'Kurucular.Ev' does not contain a definition for 'Ev' ...
```

Bu durum, şimdiye dek öğrendiklerimizle çelişiyor görünebilir. Ama unutmayalım ki, `Ev` sınıfı içinde tanımladığımız `Ev()` kurucusu genel anlamda bir metot değildir. O özel bir işlevi olan bir metottur, o bir

kurucudur. C# bir sınıf içinde o sınıfın adını alan bir metodu (kurucuyu) başkasının çalıştırmasına asla izin vermez. Sınıfa ait bir nesne kurulur kurulmaz, kurucu kendiliğinden çalışır ve üstüne yüklenen görevi sessizce yapar.

Peki ama, böyle gizemli bir metoda neden gerekseme duyarız? Onun yaptığı işi yapacak apaçık bir metot yazsak olmaz mı?

Elbette olur, ama daha zahmetli olur. Kendiliğinden çalışan kodlar bizim o an ilgilenmediğimiz ama yapılması gereken işleri yaparlar. Örneğin, programı açtığımızda sistemin giriş/çıkış birimlerinin çalışıp çalışmadığını denetleyebilir, uzaktaki bir makinaya bağlanıyorsak ağın çalışıp çalışmadığını denetleyebilir, bir veri tabanına bağlantının olup olmadığını denetleyebilir, v.b.

Kurucuyu biraz daha yakından tanımaya çalışalım. Yukarıdaki programda `Ev()` kurucusunun `public` erişim nitelemesini kaldıralım. Bunu kaldırdığımızda, `Ev()` kurucusu gendeğer (default) olarak `private` erişim belirtecini almış olacaktır. O belirteci ayrıca yazmamıza gerek yoktur.

Kurucular06.cs

```
using System;

namespace Kurucular
{
    class Uygulama
    {
        static void Main()
        {
            new Ev();
        }
    }

    class Ev
    {
        int kapıNo = 123;
        string sokakAdı = "Menekşe";

        Ev()
        {
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }
    }
}
```

Bu programı derlemek istersek, şu hata iletisini alırız:

```
Error   1   'Kurucular.Ev.Ev()' is inaccessible due to its protection level   ...
```

Bu iletiden anlıyoruz ki, `private` nitelemesi olan kurucuyu başkası çalıştıramaz; yani kurucusu `private` olan bir nesneyi başkası yaratamaz.

Şimdi merak edilen bir soruya yanıt arayalım. *"Kurucu özel bir metottur, ama hiçbir değer alamaz"* dedik. Oysa, daha önce tanımladığımız metotlar `int, string, bool, void` gibi bilinen tiplerden birinden bir değer alıyordu; yani fonksiyonu çağırdığımızda bize bir değer veriyordu ((void olsa bile bir değer sayılır). Acaba kurucu bir değer alamaz mı? Alamayacağını göstermek için, yukarıdaki programda `public Ev()` başlığı yerine

```
public void Ev()
```

başlığını yazalım ve derlemeyi deneyelim. Şu hata iletisini alacağız.

```
Error   1   'Ev': member names cannot be the same as their enclosing type   ...
```

Başlıkta `void` yerine `short, int, bool, string` gibi istediğiniz veri tipini koyunuz. Her seferinde aynı hata iletisini alacaksınız. Bunun nedeni açıktır. `public void Ev()` deyimini, derleyici bir kurucu değil, bir metot bildirimi imiş gibi algılıyor. O metodun adı ile sınıfın adının aynı olmasını kabul etmiyor. Çünkü, bir sınıf içinde kurucu dışında hiçbir öğe sınıfın adını alamaz.

Peki kurucu başka ad alabilir mi? Alamayacağını deneyerek görebiliriz. Başlığa `public Ev()` yerine

```
public  Salon()
```

yazalım ve gövdesini aynen bırakalım. Derleyiciden şu iletiyi alırız:

```
Error    1        Method must have a return type ...
```

Böyle olması doğaldır, çünkü derleyiciye bir metot bildirimi yapıyoruz. Kurucu dışındaki her metodun bir değer kümesi (veri tipi) olmak zorundadır.

Parametreli Kurucular

Kurucular özel tip metotlar olduğuna göre, onların da parametrelere bağlı olarak tanımlanabileceklerini tahmin edebiliriz. Gerçekten böyle olduğunu aşağıdaki örnekten görebiliriz.

Kurucular07.cs

```
using System;

namespace Kurucular
{
    class Uygulama
    {
        static void Main()
        {
            new Ev(123);
        }
    }

    class Ev
    {
        int kapıNo ;
        string sokakAdı = "Menekşe";

        public Ev(int n)
        {
            kapıNo = n;
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }
    }
}
```

Çıktı

```
ADRES: Menekşe Sokak, No: 123
```

Kurucu başlığını

```
public Ev(int n)
```

biçiminde yazdık. Bu sözdizimi metot bildirimindeki parametre tanımına benziyor. Bu başlık derleyiciye `Ev()` kurucusunun int tipinden n adlı bir parametresi olduğunu bildirir. `Main()` metodu, kurucuyu

```
   new Ev(123);
```

deyimiyle çağırıyor. Demek ki `n` parametresi yerine 123 değerini koydu. Bu tamamen metot çağırmaya benziyor. Kurucunun gövdesine bakarsak, `kapıNo` yerine 123 değerini atadığını ve onu konsola yazdığını görürüz.

İpucu

Kurucu başlığındaki `public Ev(int n)` yerine `public Ev()` yazılırsa, `Main()` 'in gövdesindeki `new Ev(123)` deyimi nesneyi kuramaz.

Derleyici şu hatayı iletir:

```
Error   1   'Kurucular.Ev' does not contain a constructor that takes '1' arguments   ...
```

Bu iletinin anlamı şudur. `Main()` metodu kurucuyu `Ev(123)` diye çağırdı; yani kurucunun 123 değerini alabilecek bir parametreye bağlı olduğunu varsaydı. Oysa, `Ev` sınıfı içindeki kurucu tanımı `public Ev()` şeklinde parametresizdir. `Main()` metodu 1 parametreli kurucu çağırıyor, ama öyle bir kurucu yok.

Tersine olarak, kurucu başlığında `public Ev(int n)` yazılı iken `Main()` 'in gövdesindeki `new Ev()` deyimi nesneyi kuramaz. Derleyici şu hatayı iletir.

```
Error   1   'Kurucular.Ev' does not contain a constructor that takes '0' arguments   ...
```

Bu iletinin de anlamı şudur. `Main()` metodu kurucuyu `Ev()` diye çağırdı; yani kurucunun parametresiz olduğunu varsaydı. Oysa, `Ev` sınıfı içindeki kurucu tanımı `public Ev(int n)` şeklinde 1 parametrelidir. `Main()` metodu parametresiz kurucu çağırıyor, ama öyle bir kurucu yok.

Böyle olduğunu deneyerek görünüz.

Metot çağırma kuralından biliyoruz ki, metodu çağırırken, onun bildiriminde belirtilen parametrelerin sayısı, sırası ve tipi çağrıda aynen yer almalıdır.

Şimdi iki parametreli bir kurucu tanımlayalım. Kurucu başlığına public `Ev(int n , string s)` ve çağrısına da new `Ev(123,"Menekşe")` yazalım.

Kurucular08.cs

```
using System;

namespace Kurucular
{
    class Uygulama
    {
        static void Main()
        {
            new Ev(123, "Menekşe");
        }
    }

    class Ev
    {
        int kapıNo;
        string sokakAdı;

        public Ev(int n, string s)
        {
            kapıNo = n;
            sokakAdı = s;
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }
```

```
    }
}
```

Çıktı

```
ADRES: Menekşe Sokak, No: 123
```

Yukarıdaki kodları çözümlemek kolaydır.

Kurucu başlığını

```
public Ev(int n, string s)
```

biçiminde yazdık. Bu sözdizimi metot bildirimindeki parametre tanımına benziyor. Bu başlık derleyiciye Ev() kurucusunun int tipinden n adlı bir parametresi ile string tipinden s adlı bir parametresinin olduğunu bildirir. Main() metodu, kurucuyu

```
new Ev(123, "Menekşe");
```

deyimiyle çağırıyor. Demek ki `n` parametresi yerine 123 değerini, s parametresi yerine "Menekşe" değerini koydu. Bu iş, iki parametreli metot çağırmaya benziyor. Kurucunun gövdesine bakarsak, `kapıNo` yerine 123 değerini, `sokakAdı` değişkenine "Menekşe" değerini atadığını ve onları konsola yazdığını görürüz.

Şimdiye kadar parametresiz, bir parametreli ve iki parametreli kurucular tanımladık. Metotlardaki kurala uyarak, kurucunun parametre sayısını istediğimiz kadar artırabiliriz.

Aşkın Kurucular (overloaded constructors)

Şimdi yeni bir soruya yanıt arayalım. Acaba bir sınıfın birden çok kurucusu olabilir mi?

Bu sorudaki ince nokta şudur. Metot imzası (başlığı) metodu belirleyen bütün öğelere sahiptir: değer-tipi, adı ve parametreleri. Bir metodun kurucu olabilmesi için değeri olmayacak ve ait olduğu sınıfın adını alacak. O zaman, birden çok kurucu tanımlamak için, elimizde tek seçenek kalıyor: parametrelerin sayısını, sırasını ve tipini değiştirmek. Gerçekten, parametrelerin, sayısını, sırasını ve tiplerini değiştirerek bir sınıf içinde istediğimiz kadar kurucu tanımlayabiliriz. Bu işi yaparken parametrelere yüklendiğimiz için, farklı parametrelerle birden çok kurucu tanımlama işine *aşırı yükleme (overloading)* deniyor. Aşırı yüklenmiş kurucuya da *aşkın kurucu (overloaded constructor)* diyeceğiz. Aşırı yüklemeye basit bir örnek verebiliriz.

Kurucular09.cs

```
using System;

namespace Kurucular
{
    class Uygulama
    {
        static void Main()
        {
            new Ev();
            new Ev(456);
            new Ev(789, "Papatya");
        }
    }

    class Ev
    {
        int kapıNo = 123;
        string sokakAdı = "Menekşe";

        public Ev()
        {
```

```
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }

        public Ev(int n)
        {
            kapıNo = n;
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }

        public Ev(int n, string s)
        {
            kapıNo = n;
            sokakAdı = s;
            Console.WriteLine("ADRES: {1} Sokak, No: {0} ", kapıNo,
sokakAdı);
        }
    }
}
```

Çıktı

```
ADRES: Menekşe Sokak, No: 123
ADRES: Menekşe Sokak, No: 456
ADRES: Papatya Sokak, No: 789
```

Programı çözümlemek kolaydır. `Ev` sınıfında parametresiz, 1 parametreli ve 2 parametreli olmak üzere üç kurucu tanımlandı. `Main()` metodu bu üçünü sırayla çağırdı. Her kurucu konsola bir adres yazdı. Parametresiz kurucu, adresteki `kapıNo` ve `sokakAdı` değişkenlerinin değerini sınıf değişkeninden aldı. Bir parametreli kurucu, `kapıNo` değişkeninin değerini çağrıdan, `sokakAdı` değişkeninin değerini sınıftan aldı. İki parametreli kurucu, `kapıNo` ve `sokakAdı` değişkenlerinin değerlerini çağrıdan aldı.

Aşağıdaki örnek kompleks sayı gösteren üç tane farklı kurucu tanımlamaktadır.

Kurucular10.cs

```
using System;
class Kompleks
{
    public Kompleks(int i, int j)
    {
        Console.WriteLine("{0} + i{1}" , i , j);
    }
    public Kompleks(double i, double j)
    {
        Console.WriteLine("{0} + i({1})", i, j);
    }
    public Kompleks()
    {
        int i = 5;
        int j = 8;
        Console.WriteLine("{0} + i{1}", i, j);
    }
}
class Uygulama
{
    public static void Main()
    {
```

```
        Kompleks c1 = new Kompleks(20, 25);
        Kompleks c2 = new Kompleks(2.5, 5.9);
        Kompleks c3 = new Kompleks();
    }
}
```

Statik Kurucular

Statik kurucu C# ile ortaya çıkan yeni bir kavramdır. Sınıfa ait ilk dinamik nesne yaratılmadan önce statik kurucu çağrılır. Sözdizimi şöyledir:

Kurucular11.cs

```
public class Deneme
{
    static Deneme()
    {
        // Başlatma kodları.
        // Yalnız statik öğelere erişilir
    }
    // Sınıfın öteki metotları
}
```

Özellikler:

1. Sınıfın bir tek statik kurucusu olabilir.
2. Statik kurucu parametresizdir.
3. Sınıfın ancak statik öğelerine erişebilir.
4. Statik kurucunun erişim belirteci olmaz.

Aşağıdaki program statik kurucuya basit bir örnektir.

Kurucular12.cs

```
using System;
class basitSınıf
{
    static long tikTak;
    static basitSınıf()
    {
        tikTak = DateTime.Now.Ticks;
    }
    public static void Main()
    {
        new basitSınıf();
        Console.WriteLine(basitSınıf.tikTak);
    }
}
```

Özet

Kurucu çağrılınca sınıfa ait bir nesne yaratılır; nesne yaratılır yaratılmaz kurucunun kodları kendiliğinden çalışır.

Eğer sınıfın bir kurucusu yoksa, CLR (Common Language Runtime) kendiliğinden sınıfa ait bir nesne yaratır. Buna *genkurucu* (default constructor) denilir.

Bir sınıfın istenildiği kadar kurucusu olabilir. Bir kurucunun parametrelerinin tipi, sayısı, sırası değişince farklı bir kurucu elde edilir *(Aşkın Kurucu)*.

- Kurucular değer almaz
- Kurucular aşırı yüklenebilirler.
- Bir sınıfta statik ve dinamik kurucular varsa, öncelik dinamik kuruculardadır.

Yokediciler (Destructors)

Birinci Bölümün sonunda açıkladığımız gibi, C# dili işi biten nesneleri Çöp Toplayıcı (GC- Garbage Collection) ile bellekten atar, boşalan bellek bölgesini Heap 'e ekler. Java ve C# dillerinde otomatik yapılan bu iş C++ dilinde programcı tarafından yapılır; yani programcı yarattığı bir nesneyi işi bitince bellekten silecek kodu da yazmak zorundadır.

Çöp toplayıcı bellekten işi biten nesneleri atmakta kusursuzdur, ama atma zamanını programcı belirleyemez; o derleyicinin işidir. Bu nedenle, çok özel durumlarda, C# da yaratılan bir nesnenin çöp toplayıcının keyfine bırakılmadan, işi biter bitmez bellekten silinmesi istenebilir. Bunu yapmak için kurucu'nun (constructor) tersine iş yapan yokedici (destructor) kullanılır.

Yokedici sınıf içinde sınıf adıyla parametresiz bir kurucu gibi tanımlanır, ancak önüne (~) simgesi konulur. Sözdizimi şöyledir:

```
class Ev
{
    ~ Ev()   // yokedici
    {
            // silme deyimleri
    }
}
```

Yokediciler yalnızca sınıflar içindir; yapılarda kullanılmaz.

Bir sınıfın yalnızca bir tane yokedicisi olabilir.

Yokedici aşırı yüklenemez, kalıtımı olamaz.

Yokedici çağrılamaz; o kendisi otomatik çalışır.

Yokedici değiştirilemez; parametresi yoktur.

Alıştırmalar

1. Aşağıdaki programdaki hatayı bulup düzeltiniz.

Kurucular13.cs

```
class Uygulama
{
    static void Main()
    {
        System.Console.WriteLine(birSınıf.i);
    }
}
```

```
class birSınıf
{
    public int i = 10;
}
```

2. Aşağıdaki programı koşturmadan çıktıyı tahmin ediniz.

Kurucular14.cs

```
class Uygulama
{
    static void Main()
    {
        birSınıf a = new birSınıf();
        birSınıf b = new birSınıf();
        a.j = 11;
        System.Console.WriteLine(a.j);
        System.Console.WriteLine(b.j);
        birSınıf.i = 30;
        System.Console.WriteLine(birSınıf.i);
    }
}
class birSınıf
{
    public static int i = 10;
    public int j = 10;
}
```

3. Aşağıdaki programı koşturmadan çıktıyı tahmin ediniz.

Kurucular15.cs

```
class Uygulama
{
    public static void Main()
    {
        birSınıf a;
        System.Console.WriteLine("Main");
        a = new birSınıf();
    }
}
class birSınıf
{
    public birSınıf()
    {
        System.Console.WriteLine("birSınıf");
    }
}
```

3. Aşağıdaki programı çözümleyiniz.

Kurucular16.cs

```
using System;
public class Üçgen
{
    private int _yükseklik;
    private int _taban;
```

```
    public int Yükseklik
    {
        get
        {
            return _yükseklik;
        }
        set
        {
            if (value < 1 || value > 100)
                throw new OverflowException();

            _yükseklik = value;
        }
    }

    public int Taban
    {
        get
        {
            return _taban;
        }
        set
        {
            if (value < 1 || value > 100)
                throw new OverflowException();

            _taban = value;
        }
    }

    public double Alan
    {
        get
        {
            return _yükseklik * _taban * 0.5;
        }
    }


    public Üçgen()
    {
        Console.WriteLine("Üçgen kurucusu çalıştı");

        _yükseklik = _taban = 1;
    }
}

class Uygulama
{
    static void Main(string[] args)
    {
        Üçgen üçgen = new Üçgen();

        Console.WriteLine("Yükseklik:\t{0}", üçgen.Yükseklik);
        Console.WriteLine("Taban:\t{0}", üçgen.Taban);
        Console.WriteLine("Alan:\t{0}", üçgen.Alan);
    }
}
```

4. Şimdi yukarıdaki kurucuyu parametreli hale getirelim.

Kurucular17.cs

```
using System;
public class Üçgen
{
    private int _yükseklik;
    private int _taban;

    public int Yükseklik
    {
        get
        {
            return _yükseklik;
        }
        set
        {
            if (value < 1 || value > 100)
                throw new OverflowException();

            _yükseklik = value;
        }
    }

    public int Taban
    {
        get
        {
            return _taban;
        }
        set
        {
            if (value < 1 || value > 100)
                throw new OverflowException();

            _taban = value;
        }
    }

    public double Alan
    {
        get
        {
            return _yükseklik * _taban * 0.5;
        }
    }


public Üçgen(int yükseklik, int taban)
{
    Console.WriteLine("Üçgen kurucusu çalıştı");

    this.Yükseklik = yükseklik;
    this.Taban = taban;
}
}

class Uygulama
{
```

```
    static void Main(string[] args)
    {
        Üçgen üçgen = new Üçgen(7,9);

        Console.WriteLine("Yükseklik:\t{0}", üçgen.Yükseklik);
        Console.WriteLine("Taban:\t{0}", üçgen.Taban);
        Console.WriteLine("Alan:\t{0}", üçgen.Alan);
    }
}
```

5. Aşağıdaki programı güneşin yedi gezegenini kapsayacak şekilde değiştiriniz.

Kurucular18.cs

```
using System;
class Gezegen
{

    public int yarıÇap;
    public double yerÇekim;
    private string ad;
    private static int sayaç;

    public Gezegen(int r, double g, string n)
    {
        yarıÇap = r;
        yerÇekim = g;
        ad = n;
        sayaç++;
    }

    public string adYaz()
    {
        return ad;
    }

    public static int GezegenSay()
    {
        return sayaç;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Gezegen earth = new Gezegen(6378, 9.81, "Earth");
        Gezegen saturn = new Gezegen(60268, 8.96, "Saturn");
        Console.WriteLine("Gezegen Sayısı: " + Gezegen.GezegenSay());
    }
}
```

Bölüm 05

Veri Tipleri ve Değişkenler

C# Dilinde Veri Tipi nedir?

C# dilinde her sınıf bir veri tipidir, her veri tipi bir sınıftır.

Klâsik dillerde *tamsayılar, kesirli sayılar, karakterler, boolean değerler* gibi ilkel veri tipleri dile gömülü öğelerdir; herbiri bir anahtar sözcükle belirtilir. Oysa, Nesne Yönelimli Programlamada her sınıf soyut bir veri yapısıdır. O nedenle, C# dilinde klâsik dillerdeki gibi ilkel veri tipleri yoktur. Her veri tipi bir sınıftır. Ancak, programcının işini kolaylaştırmak için, .NET bu sınıfları ve sınıf öğelerini hazır tanımlamıştır. .NET Framework ortamını anlatırken *C++, C#, VB, J#* dillerinin temel veri tiplerinin .NET'in *CTS* (Common Type System) denilen veri tiplerine dönüştüğünü, böylece bu diller için ortak bir platform yaratılmış olduğunu söylemiştik. C# dilinin temel veri tiplerinin .NET 'deki karşılıklarını *Veri Tipleri ve Değişkenler* bölümünde ayrıntılı olarak ele alacağız. Bu sınıflar sanki klâsik dillerdeki ilkel veri tipleriymiş gibi kullanılabilirler. Aşağıda ayrıntıları verilecek olan *temel veri tiplerinin (built-in-types)* birer sınıf olduğunu, dolayısıyla klâsik dillerdeki ilkel veri tiplerine göre çok daha işlevsel olduklarını daima anımsamalıyız.

Neden Veri Tipi?

Bir programda farklı veri tipleriyle işlem yapmamız gerekebilir. Örneğin, tamsayılar, kesirli sayılar, karakterler (harfler ve klavyedeki diğer simgeler), metinler (string), mantıksal (boolean) değerler (doğru=true, yanlış=false) ilk aklımıza gelen farklı veri tipleridir. Bu farklı veri tiplerinin büyüklükleri (bellekte kaplayacakları yer) ve onlarla yapılabilecek işlemler birbirlerinden farklıdır. Örneğin, sayılarla dört işlem yapabiliriz, ama metinlerle yapamayız. O nedenle, C# ve başka bazı diller verileri tiplere ayırır. Değişken tanımlarken onun hangi tip veriyi tutacağını belirtir. Böylece, ana bellekte o değişkene yetecek bir yer ayırır ve o veri tipine uygun işlemlerin yapılmasına izin verir.

Değişken Nedir?

Teknik açıdan, değişken, ana bellekte belli bir veri tipine ait değerlerin girilebileceği bir adrestir.

Değişkenler programın ham veri tiplerini taşıyan araçlardır. C# dilinde global değişken yoktur. Her değişken bir sınıf içinde ya da sınıftaki bir blok içinde tanımlıdır. Sınıf içinde tanımlı olanlar tanımlandığı sınıfın bir öğesi (class member) olur. Sınıf içindeki bir blok içinde tanımlananlar o blokun yerel değişkeni olur.

Her değişkene, ana bellekte, o değişkenin tutacağı veri tipine yetecek büyüklükte bir yer ayrılır. Bu yere ana bellekte *değişkenin adresi* denir. Her değişkene bir ad verilir. Bu ad kullanılarak, değişkene değer atanabilir, atanan değer okunabilir ve atanan değer değiştirilebilir (güncellenebilir). Bir değişkene erişim demek, o değişkene değer atama, atanan değeri okuyabilme ve atanan değeri istendiğinde değiştirebilme yeteneklerine sahip olmak demektir.

C# *tip-korumalı* bir dildir. Başka bir deyişle, C# dili programda kullanılacak değişkenlerin tutacağı veri tiplerini kesin sınırlarla birbirlerinden ayırır. Bu yönüyle C, C++ ve Java'ya benzer. Dolayısıyla, bir değişken bildirimi yapılırken, o değişkenin veri tipi kesinlikle belirtilir. Daha sonra o değişkene atanan veriler, belirtilen veri tipinin dışına çıkamaz. Bunun bir istisnası, bazı durumlarda bir değişkenin tuttuğu verinin başka bir tipe dönüştürülmesidir. İngilizce'de casting denilen bu yöntemi ileride açıklayacağız.

C# dilinde değişken bildirimi aşağıdaki gibi yapılır.

```
int  faizOranı;
```

Bu deyim, `int` tipinden `faizOranı` adlı bir değişkenin bildirimidir. Bazı kaynaklarda buna değişken tanımı denilir. Biz bildirim sözcüğünü yeğlemekle birlikte, yerine göre her ikisini eş anlamlı kullanacağız. Bazı durumlarda, değişkene hemen bildirim anında bir ilk-değer vermek gerekebilir. Bu durumda,

```
int  faizOranı = 6 ;
```

yazılır. Bu bildirim int tipinden faizOranı adlı bir değişken bildirmekte ve ona ilk-değer olarak 6 tamsayısını atamaktadır. Bu söylediklerimizi genelleştirirsek, değişken bildirimi için aşağıdaki sözdizimini kullanacağız:

```
veri_tipi   değişken_adı ;
veri_tipi   değişken_adı = ilk_değer ;
```

biçimindedir. Tabii, bildirim anında ilk değer vermek zorunlu değil, isteğe bağlıdır.

Kural

Değişkene değer atarken, değişken ile değeri arasına (=) simgesi konulur. Bu nedenle (=) simgesine atama operatörü diyoruz.

C# küçük-büyük harf ayrımı yapar. Değişken adları rakamla başlayamaz ve ad içinde boşluk olamaz.

Değişken adını istediğiniz gibi koyabilirsiniz. Ama değişken adlarını küçük harfle, sınıf adlarını büyük harfle başlatmak, bir gelenektir. Bu geleneğe uyunuz. Ayrıca, programda tanımladığınız değişken, sabit, sınıf, metot gibi varlıkların adlarını, işlevlerini ima edecek biçimde koymak program okumayı kolaylaştıran iyi bir alışkanlık

olacaktır. Bazı durumlarda, adlar birden çok sözcükten oluşabilir. O zaman sözcükler arasına (_) simgesi koymak izlenen yöntemlerden birisidir. Ama C# dilinde (_) simgesini kullanmak yerine, deve-notasyonu denen yöntem tercih edilir. Bu stilde, farklı sözcüklerin ilk harfleri büyük yazılarak bitişik olarak yazılır. Örneğin, aşağıdakiler geçerli birer değişken adıdır.

```
faizOranı, sicilNo, öğrencininAdıVeSoyadı, sigortalınınDoğumTarihi
```

Aynı değişkenleri

```
faiz_oranı, sicil_no, öğrencinin_adı_ve_soyadı, sigortalının_doğum_tarihi
```

diye de yazabilirsiniz. Ama C# ilk yöntemi tercih etmektedir.

Aynı veri tipinden olan birden çok değişken bildirimini tek bir deyimle yapabiliriz:

```
int  m, n, r ;
```

Geçerlik Bölgesi

Her değişken tanımlı olduğu bölgede geçerlidir. Bunun anlamı şudur. Değişken sınıfın bir öğesi ise, bütün sınıf içinde geçerlidir. Ona sınıfın her metodu, her bloku erişebilir. Sınıfa ait nesne bellekte kaldığı sürece değişken kendi adresini korur.

Sınıftaki bir blokun içinde tanımlı olan değişken yalnız o blok içinde geçerlidir; ona tanımlı olduğu blok içinden ve o blokun alt bloklarından erişilebilir. Blokun işi bitince bellekteki adresi silinir. Örneğin bir for döngüsünün sayacı döngü bloku içinde geçerlidir. Döngü başlarken ona bellekte bir yer ayrılır, döngü bitince bellekten silinir. Benzer olarak, bir metot içinde tanımlı olan bir değişkene metot çağrılınca ana bellekte bir yer ayrılır, metodun işi bitince bellekten silinir.

Bu nedenle, bir metot, bir döngü veya {} parantezleriyle belirlenen bir blok içinde tanımlı değişkene *yerel değişken* diyoruz. Hiçbir blok içinde olmayıp sınıf içinde tanımlı değişkenlere de *sınıf değişkenleri* ya da *sınıf öğeleri* (class member) diyoruz.

İç-içe bloklar olduğunda, dış bloktaki değişkene iç bloktan erişilebilir. Ama, dış bloktan iç bloktaki değişkene erişilemez. Bir sınıf içinde tanımlı metot, döngü veya {} ile belirli bloklara göre, sınıfın kendisi de bir bloktur ve öteki blokların hepsini içine alır; yani en dıştaki bloktur. Dolayısıyla, sınıf öğelerine iç blokların her birinden erişilebiliyor olması doğaldır.

Değer Tipleri- Referans Tipleri

Birinci bölümde ana bellekteki stack ve heap bölgelerinden söz etmiştik. C# değişkenleri stack ve heap'te oluşlarına göre ikiye ayırır.

1. Değer tipleri
2. Referans tipleri

Değer tipi değişken, ana bellekteki adresine bir değer yazılan değişkendir. Bu değişkenler *stack*'ta tutulur. Referans tipi değişken ise, ana bellekteki bir nesneyi işaret eden (reference, pointer) değişkendir. İşaret edilen nesneler *heap*'te tutulur. Sınıflara ait bütün nesneler heap'te tutulur.

C# Dilinde sabit nedir?

Sabitler de değişkenler gibi veri tutan sınıf öğeleridir. Dolayısıyla her sabite, ana bellekte tutacağı veri tipine yetecek kadar bir yer ayrılır. Bunun değişkenden tek farkı şudur: Değişkenlere atanan değerler program çalışırken değiştirilebilir (güncelleme). Ancak, sabitlere atanan değerler değiştirilemez. Ayrıca, sabit bildirimi yapıldığı anda değeri atanmalıdır. Örneğin,

```
const  int  yıllıkFaizOranı = 8 ;
const  float  amortismanPayı = 12.50;
```

Genel söz dizimi şöyledir:

```
const  veri_tipi  sabitin_adı = atanan_değer ;
```

Sabit kullanmadan, sabitin işleme girdiği her yerde onun değerini koyarak program yazmak mümkündür. Örneğin, yukarıda bildirimi yapılan yıllıkFaizOranı bir bankanın mudilerinin alacağı faizleri hesaplayan programda kullanılıyor olsun. Yıllık faizin hesaplandığı her işlemde yıllıkFaizOranı yerine 8 kullanılabilir ve program aynı sonucu verir. Ancak faiz oranını değiştirip 9 yapmak istediğinizde, kaynak programın bütününde 8 yerine 9 yazmak gerekecektir. Bu hem uzun zaman alıcı hem de bazıları unutulabileceği için, hataya açıktır. Onun yerine yıllıkFaizOranı adlı bir sabit kullanılırsa, kaynak programda onun değerini 9 ile değiştirmek, bütün programı güncellemeye denktir. Bunu yapmak büyük bir zaman kaybını önleyeceği gibi, programda hata oluşmasının da önüne geçmiş olacaktır.

Sayısal Değişken Tipleri

Bir çok işlemi tamsayılarla ve kesirli sayılarla yaparız. O nedenle, sayılar her programlama dilinde önemlidirler. Farklı dillerde yazılan derleyiciler sayıları alt sınıflarına ayırır. En basit derleyiciler bile, sayıları tamsayılar (integer) ve kesirli sayılar (floating numbers) diye ikiye ayırır. Bazan küçük sayılarla, Bazan da büyük sayılarla uğraşırız. Dolayısıyla, belleği etkin kullanabilmek için, bir çok dilde, sayılar, alt-gruplarına ayrılır. C# dili, bu yönde çok ayrıntılı bir gruplama yapmıştır.

Tamsayı Değişken Tipleri (Integer Variable Types)

C# dili tamsayıları bellekte kendilerine ayrılan büyüklüğe ve işaretli olup olmadıklarına göre gruplara ayırır. Aşağıdaki tabloda bu grupları, .NET karşılıklarını ve özeliklerini göreceksiniz.

C# Tipi	.NET Tipi	Uzunluk (Byte)	Değer Aralığı
byte	Byte	1 byte	0 - 255
sbyte	SByte	1 byte	-128 - 127
short	Int16	2 byte	-32 768 - 32 767
ushort	UInt16	2 byte	0 - 65 535
int	Int32	4 byte	-2 147 483 648 - 2 147 483 648
uint	UInt32	4 byte	0 - 4 294 967 295
long	Int64	8 byte	-10^{20} - 10^{20}
ulong	UInt64	8 byte	0 - 2×10^{20}

C# dilinde tamsayı veri tipleri

İpucu

C# dilinin veri tiplerinin .NET 'in veri tiplerine dönüştüğünü söylemiştik. Dolayısıyla, kaynak programlarımızda C# veri tipleri yerine .NET 'teki karşılığını kullanabiliriz. Örneğin,

```
short n ;
int m ;
long r ;
```

bildirimleri yerine, sırasıyla,

```
Int16 n;
Int32 m ;
Int64 r ;
```

bildirimlerini yapabiliriz. C# derleyicisi bunları denk deyimler olarak algılar.

Kesirli Sayı Değişken Tipleri (Floating Point Variables)

C# dilinde kesirli sayılar float ve double diye ikiye ayrılır. Aşağıdaki tabloda bu grupları ve özeliklerini göreceksiniz. Aksi istenmediğinde, C# kesirli sayıların double tipinden olduğunu varsayar (default tip).

C# Tipi	.NET Tipi	Uzunluk (Byte)	Değer Aralığı	Duyarlı Basamak Sayısı
float	Single	4 byte	$1.5 * 10^{-45}$ - $3.4 * 10^{38}$	6 - 7 basamak
double	Double	8 byte	$5.0 * 10^{-324}$ - $1.7 * 10^{308}$	15 - 16 basamak

C# dilinde kesirli sayı veri tipleri

Örnek

C# kesirli sayıları, aksi söylenmediği zaman `double` tipinden sayar. Bu öndeğer (default) durum istenmiyorsa, kesirli sayının sonuna `f` ya da `F` yazılır. Aşağıdaki örnek, `float` veri tipinin kullanılışını göstermektedir

VeriTipi01.cs

```
using System;
namespace Bölüm05
{
    class KarmaTipler01
    {
        public static void Main()
        {
            int x = 3;
            float y = 4.5f;
            short z = 5;
            Console.WriteLine("{0} * {1} / {2} = {3}", x,y,z,x * y/z);
        }
    }
}
```

Bu programın çıktısı:

```
3 * 4,5 / 5 = 2,7
```

Çıktıda kesir kısmını 3 haneli yazdırmak için `f3` biçemleyicisini kullanırız. (Sayısal çıktıları biçemlemeyi ileride ele alacağız.)

VeriTipi02.cs

```
using System;

namespace Bölüm05
{
    class KarmaTipler02
    {
        public static void Main()
        {
```

```
            int x = 4;
            float y = 2.5f;
            float z = 5.0f;
            Console.WriteLine("{0} * {1} / {2} = {3:f3}", x,y,z,x*y/z);
        }
    }
}
```

Çıktı

```
4 * 2,5 / 5 = 2,000
```

Decimal Veri Tipi

C# dili tamsayıları ve kesirli sayıları, yukarıdaki tablolarda görülen alt-gruplarına ayırmıştır. Bu gruplar belleğin optimal kullanımını sağlar. Ancak, her gruba özgü sınırlamalar vardır. Örneğin, özellikle tip dönüşümü (casting) istenmezse, iki tamsayının birbirine bölümünden çıkan bölümün kesir kısmı atılır, yalnızca tamsayı kısmı yazılır. Dolayısıyla, bölüm kesirli sayı olması gerekirken, tamsayı olur. Kesirli sayı tiplerinde ise, yuvarlamalar nedeniyle kayıplar olabilir. Bu iki sorunu çözmek için C# decimal veri tipini yaratmıştır.

C# Tipi	.NET Tipi	Uzunluk (Byte)	Değer Aralığı	Duyarlı Basamak Sayısı
decimal	Decimal	12 byte	$\pm 1.0 \times 10^{-28}$ - $\pm 7.9 \times 10^{28}$	28 - 29 basamak

C# dilinde decimal sayı veri tipi

Char Veri Tipi

Character sözcüğü, kaynak programda kullanılan bütün harf, rakam ve diğer simgelerin kümesidir. C# dili, karakterleri unicode diye adlandırılan ve uzunluğu 2 byte (16 bit) olan sistemle belirler. Bu nedenle 1 byte (8 bit) lık sistemlerde olduğu gibi yalnızca İngiliz alfabesini değil, Türkçe, Arapça, Çince, Japonca, Rusça, İbranice vb. bütün dillerin alfabelerindeki harfleri içine alır.

Burada şuna dikkat çekmeliyiz. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 rakamların her birisi sayısal değerlerinin dışında, ayrıca birer character olarak character veri tipinde yer alırlar. Zaten bütün sayıların ekrana ya da kağıda gönderilen görüntüleri 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 karakterleriyle temsil edilirler.

C# Tipi	.NET Tipi	Uzunluk (Byte)	Kapsam
char	Char	2 byte	Bütün unicode karakterleri

C# dilinde char veri tipi

C# dilinde character veri tipi, kısaca char diye yazılır. Bu tipli bir değişken bildirimi aşağıdakilere benzer olarak yapılır:

```
char  birHarf  ;
```

ya da bildirim anında ilk değeri verilmek istenirse

```
char  birHarf  = 'K' ;
```

İpucu

`char` değerler daima tek tırnak (‘ ‘) içinde yazılır. Örneğin,

```
'b' , '0' , '7' , '$' , '=' , '?' , '''
```

birer karakterdir, ama derleyici

```
b , 0 , 7 , $ , = , ? , '
```

simgelerini birer karakter olarak algılamaz.

Bütün dillerde olduğu gibi, C# dili de klavyede tek bir simgeyle temsil edilemeyen bazı karakterleri kullanır. Bunlar, özellikle, yazıcıya ya da ekrana çıktı alırken kullanılan karakterlerdir. Bazı harflerin önüne (\) konularak yazılırlar, ama derleyici onları birer karakter olarak algılar:

```
\n    New Line (yeni satıra geç)
\t    Tab (tab yaz)
\0    Null (boş)
\r    Carriage Return (satırbaşına git)
\\    Backslash (ters bölü çizgisi (\) yazar)
```

String Veri Tipi

`string` veri tipi, `String` sınıfının takma adıdır. Tek bir karakter yerine bir sözcük, bir tümce, bir paragraf ya da bir roman yazmak istediğimizde char veri tipi yeterli olmayacaktır. Modern dillerin çoğunda olduğu gibi, C# dili de metinleri içerecek bir veri tipi yaratmıştır. Bu tipe string denilir. string veri tipi (sınıfı), uzun ya da kısa her metni içerebilir. Tabii, burada metin derken, yalnızca alfabenin harf, rakam ve işaretleriyle yetinmiyor, char tiplerinden oluşabilecek her diziyi kastediyoruz.

String tipleri çift tırnak (" ") içine yazılırlar. char tipler tek tırnak (' ') içine yazıldığı için, C# derleyicisi

```
'b'
```

verisini char olarak algılar, ama

```
"b"
```

verisini string olarak algılar.

C# dilinde string tipli bir değişken bildirimi aşağıdakilere benzer olarak yapılır:

```
string birMetin  ;
```

ya da bildirim anında ilk değeri verilmek istenirse

```
string birMetin = "Merhaba, dünya!" ;
```

İpucu

string değerler daima çift tırnak (" ") içinde yazılır. Örneğin,

```
"b" , "0" , "Ankara" , "&%$*/?" , "3 + 2" , "123" , "-123"
```

birer string'dir. Ama, ama derleyici

```
b , 0 , Ankara , &%$*/? , 3 + 2 , 123 , -123
```

simgelerini birer string olarak algılamaz.

Uyarı

C# dilinde string değeri birden çok satıra yazılamaz. Metin satıra sığmadığı ya da satırlara bölmek gerektiğinde, yeni satıra geçmeyi sağlayan (\n) karakteri kullanılır. Örneğin,

```
string  birŞiir = "Neler yapmadık bu vatan için \n Kimimiz öldük, \n Kimimiz nutuk söyledik. \n O.V.Kanık";
```

Bunu ekrana yazdırmak için

```
System.Console.WriteLine (birŞiir);
```

metodunu kullanırsak, ekrana şu görüntü gelir:

Neler yapmadık bu vatan için

Kimimiz öldük,

Kimimiz nutuk söyledik.

O.V.Kanık

Boolean Veri Tipi

Matematiksel Mantığın kurucusu sayılan George Boole'un adına izafe edilen ve Boolean diye adlandırılan mantıksal veri tipi iki tanedir: `true (doğru), false (yanlış)`. Bütün programlama dillerinde boolean veri tipi çok önem taşır. Özellikle, program akışının yönlendirilmesinde ve döngü yapılarında olmazsa olmaz tiplerdir.

`bool` anahtar sözcüğü `System.Boolean` 'ın takma adıdır.

C# Tipi	.NET Tipi	Uzunluk (Byte)	Kapsam
bool	Boolean	1 byte	true , false

C# dilinde bool veri tipi

`bool` tipi değişken bildirimi, genel değişken bildirimi kuralına göre yapılır:

```
bool  değişken_adı;
```

ya da ilk değer atanmak istenirse

```
bool  değişken_adı = değişken_değeri;
```

yazılır. Bazı durumlarda true ya da false literal değerleri yerine, bir mantıksal (boolean) deyim de atanabilir. Örneğin,

```
bool  mezun;
bool  mezun   = true;
bool  emekli  = false ;
bool  bFormül = (x > 15 && x < 30);
```

VeriTipi03.cs

```
using System;

namespace Bölüm05
{
    using System;
    class test
    {
        public static void Main()
        {
            bool a = true;
            Console.WriteLine(a ? "evet" : "hayır");
        }
    }
}
```

Veri Tipi Dönüşümleri (Casting)

Veri tiplerini, bellekte kendilerine ayrılan yerin byte cinsinden uzunluğuna göre gruplamıştık. Bazan sayısal bir değişkenin değerini bir tipten başka bir tipe dönüştürmemiz gerekebilir. Bu eylem, veriyi bellekte bir adresten başka bir adrese taşımak demektir. Öyleyse, şöyle bir akıl yürütebiliriz. Daha küçük bir yere sığan bir veri daha büyük bir yere taşınabilir. Ama daha büyük bir yere sığan bir veri daha küçük bir yere taşınamaz. Taşınırsa, veri kaybı olabilir.

İstemsiz (implicit) Dönüşüm

Bunu bir örnekle açıklamak daha kolay olacaktır. Anımsayacağınız gibi tamsayı veri tiplerini sekiz alt gruba bölmüştük. Onlar arasında `int` tipi değişkenler ana bellekte 4 byte (32 bit), `long` tipi değişkenler ise ana bellekte 8 byte (64 bit) uzunluğunda bellek adreslerine sahiptirler.

```
int  aaa = 123;
long  bbb ;
```

bildirimlerini yapmış olalım. Sonra,

```
bbb = aaa ;
```

atamasını yaparsak, aaa 'nın bellekteki adresinde yer alan 123 sayısını bbb 'nin bellekteki adresine taşımış oluruz. Bu taşımayı 4 byte'lık bir yerden 8 byte'lık bir adrese yaptığımız için, veri kaybı söz konusu değildir. Dolayısıyla, yukarıdaki veri tipi dönüşümü her zaman geçerlidir. Derleyici hiç itiraz etmeden dönüşümü yapan kodları derler. Bu tür dönüşüme istemsiz (implicit) dönüşüm denir. Zorlayıcı bir istem (talep) olmadan, yazılan dönüşüm deyimine derleyici itirazsız uyar.

Ama yukarıdaki dönüşümün tersi yapılamaz; çünkü 8 byte'lık bir yere sığan sayıyı 4 byte'lık bir yere sığdıramazsınız. Bunu yapmak isterseniz, C# derleyicisi izin vermez ve o dönüşümü yapan kodu derlemez.

İstemli(explicit) Dönüşüm

Bazı durumlarda long tipinden tanımlanan bir değişkene atanacak değerlerin int tipinden asla büyük olmayacağı programcı tarafından biliniyorsa, derleyiciyi ikna ederek long tipi veriyi int tipine dönüştürebilir. Buna casting denilir. Bunun nasıl yapıldığını, yukarıdakine benzer bir örnekle açıklayalım:

```
int  ccc ;
long  ddd= 456 ;
```

bildirimlerini yapmış olalım. Sonra,

```
ccc = (int)ddd ;
```

atamasını yaparsak, ddd 'nin belekteki adresinde yer alan 456 sayısını ccc 'nin bellekteki adresine taşımış oluruz. Bu taşımayı 8 byte'lık bir yerde4 byte'lık bir adrese yaptığımız için, ccc = (int)ddd atama deyiminde ddd değişkeninin önüne (int) simgesini koyuyoruz. Bunu yapmakla, derleyiciyi büyük adresteki veriyi küçük adrese taşımaya ikna ediyoruz. Bu tür dönüşüme istemli (explicit) dönüşüm denir.

Uyarı

Bu tür dönüşümler ancak sayısal veriler için yapılabilir. string ya da bool tipleri arasında yapılamaz.

Örnekler

64 bitlik bir bellek alanı işgal eden bir `double` veri tipini 32 bitlik bir `int` veri tipine dönüştürelim.

```
int  n ;
double  x  = 12.73;
```

bildirimleri yapılmış olsun.

```
n = (int)x;
```

istemli dönüşümü, x değişkeninin değeri olan 12.73 sayısının kesir kısmını atar ve 12 tamsayı kısmını `n` değişkeninin bellekteki adresine aktarır. Dönüşümde veri kaybı vardır. O nedenle, bu tür dönüşümü derleyici kendiliğinden (istemsiz - implicit casting) yapmaz. Bu dönüşüm, *istemli bir dönüşümdür* (explicit casting). Programcının bu dönüşümü özellikle isteyerek ve veri kaybı olabileceğini bilerek yapması gerekir. Bu dönüşümden sonra `n` değişkeninin değeri

```
n = 12
```

olacaktır.

Yukarıdaki dönüşümde, `double` tipinden `int` tipine geçerken, kesir kısmının atılacağını tahmin etmek zor değildi. Ama, bilinçsizce yapılan dönüşüm, beklenmedik sonuçlar doğurabilir. Şu örneği inceleyelim.

```
short sss ;
int   nnn  = 32800;
```

bildiriminden sonra

```
short  sss    =  nnn;
```

istemsiz (implicit) ve

```
short  sss    = (short) nnn;
```

istemli (explicit) dönüşümlerini istediğimizde, derleyicinin yapacağı işlere bakalım. Önce istemsiz dönüşümü yazalım.

VeriTipi04.cs

```
using System;

namespace DenetimYapıları
{
    using System;
    class test
    {
        public static void Main()
        {
            short sss;
            int nnn = 32800;
            sss = nnn;
            Console.WriteLine(sss);
        }
    }
}
```

Derleyici bu programı derlemez ve şu hata iletisini verir.

```
Error 1 Cannot implicitly convert type 'int' to 'short'. An explicit conversion exists (are you missing a cast?) ...
```

Bu demektir ki, derleyici `int` tipinden `short` tipine *istemsiz* (implicit) dönüşüm yapamaz. Çünkü, `int` veri tipi `32` bitlik bir bellek alanı tutarken, `short` tipi 16 bitlik bir yer tutmaktadır.

Şimdi de derleyiciyi bu dönüşümü yapmaya zorlayalı. Bunun için, istemsiz dönüşümü, istemliye çevirmeliyiz:

VeriTipi05.cs

```
using System;

namespace DenetimYapıları
```

```
{
    using System;
    class test
    {
        public static void Main()
        {
            short sss;
            int nnn = 32800;
            sss = (short)nnn;
            Console.WriteLine(sss);
        }
    }
}
```

Çıktı

```
-32736
```

Tabii, 32800 sayısını bir başka değişkene ayen aktarmak istediğimizde, asıl sayı yerine -32736 sayının aktarılmış olması kabul edilemez bir hatadır. İşin kötüsü, programda sözdizimi hatası olmadığı için, program derlenmiş ve çalışmıştır. Böyle bir işi, diyelim ki, on binlerce satırlık büyük bir programda yapsaydık, hatanın farkına bile varamayabilirdik.

İpucu

Derleme veya koşma anında hata mesajı vermeyen bu tür yanlışlara mantıksal (semantik) yanlışlar diyoruz. Programlamada en tehlikeli yanlışlar bunlardır; çünkü farkına varılmayabilir. Derleme ve koşma anında oluşan hatalar, programcıyı yormaktan başka kimseye zarar veremez. Programcı onları düzeltmediği sürece program çalışmayacaktır. Ama mantıksal hata içeren programlardaki hatanın farkına varılmaksızın, koşmaya devam edebilirler.

Şimdi, böylesine ölümcül bir hatanın neden oluştuğunu araştıralım. Bu mantıksal hatayı yaratan nedeni kavrarsak, bir daha bu tür hatalara düşmeyiz.

Örneğimizde `nnn` ve `sss` değişkenlerinin her ikisi de tamsayıdır, ama `short` tipinden olan `sss` değişkeninin 16 bitlik adresine -32768 ile +32767 arasındaki tamsayılar sığar. Çünkü, 16 bitlik bellekte işaretli sayılar şöyle temsil edilir:

2-li gösterimdeki en yüksek bit (en soldaki bit) *işaret bit*'idir. İşaret biti **0** ise sayı pozitif, işaret biti **1** ise sayı negatiftir. Buna göre 16-bitlik bellek adresine sığacak *en büyük pozitif tamsayının* ikili sayıtlama dizgesindeki temsili şöyledir:

0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Bunun 10-lu sistemdeki karşılığı

$$2^{14} + 2^{13} + 2^{12} + \cdots + 2^2 + 2^1 + 2^0 = \frac{1 - 2^{15}}{1 - 2} = 32767$$

olur.

Negatif sayılar bellekte *2 nin tümleyeni* denilen sistemle tutulur. Bu sistemde, işaret biti hariç, sayının öteki bitleri 0 ise 1, 1 ise 0 yapılır ve çıkan sayıya 1 eklenir. Negatif sayıların işaret biti daima 1 dir. Şimdi derleyiciyi,

```
short  sss    = (short) nnn;
```

istemli dönüşümüyle, 32800 sayısını `short` tipinden olan `sss` değişkeninin adresine yazmaya zorluyoruz. 32800 sayısının 32 bitlik bellek adresinde 2-li sayıtlama sistemine göre temsili şöyledir.

00000000 00000000 10000000 00100000

Bunu 16 bitlik adrese girmeye zorladığımızda, sağdaki 16 bit yerleşir ve dolayısıyla `sss` değişkeninin değeri

10000000 00100000

olur. İşaret biti 1 olduğu için, derleyici bunu negatif `short` veri tipi olarak algılayacaktır. Kural gereği böyle bir sayıyı *2-nin tümleyenine* dönüştürecektir. Bunu yaparken, en soldaki işaret bitine dokunmaz, öteki hanelerde 1 olanları 0 ve 0 olanları 1 yapar, sonra çıkan sayıya 1 ekler. 0 ları 1 ve 1 leri 0 yapınca, sayı

1**1**1111111 11011111

biçimini alacaktır. Bunu 1 ile toplarsak

1**1**1111111 11100000

olur. Bunun 10-lu sayıtlama sistemindeki karşılığı

$$-(2^{14} + 2^{13} + 2^{12} + \cdots + 2^6 + 2^5 + 0 + 0 + 0 + 0 + 0) = -32736$$

olur.

Buradan çıkaracağımız ders şudur. *İstemsiz (implicit)* dönüşümlerde sorun yoktur; yanlış dönüşüm yazarsak derleyici ona itiraz eder ve derlemez. Dolayısıyla program koşma anında farkında olmadığımız *mantıksal* (semantik) hatalar yapamaz. Ama *istemli (explicit)* dönüşüm derleyiciye zorla yaptırılan bir dönüşümdür. Programcı bilinçsizce istemli dönüşümler yaparsa, derleme hatası ve koşma hatası oluşmaz, ama program farkına varılamayan yanlış işlemler yapabilir. O nedenle, istemli dönüşüm yazarken, çıkacak sonucun ne olacağını biliyor ve o sonucu istiyor olmalıyız.

Alıştırma

Yukarıda söylenenleri gerçekten kavramış iseniz, aşağıdaki programı kolayca çözümler ve mantıksal hata yapıp yapmadığını bulabilirsiniz.

VeriTipi06.cs

```
using System;

class Uygulama
{
    static void Main(string[] args)
    {
        short i;
        ushort j;
        j = 60000;
        i = (short)j;
        Console.WriteLine(i);
    }
}
```

Bölüm 06

Operatörler

Aritmetik Operatörleri

Hemen her programlama dilinde olduğu gibi C# dilinde de aritmetik işlemler yaparken aşağıdaki operatörleri kullanırız:

Operatör	*Açıklama*
+	Toplama
-	Çıkarma
*	Çarpma
/	Bölme
%	Modulo
++	1 artırma
--	1 eksiltme

Bu operatörlerin kullanılışı aşağıdaki örnekte gösterilmiştir.

Operator01.cs

```
using System;

namespace Operatörler
{
    class DörtİşlemDörtİşlem
    {
        static void Main(string[] args)
        {
            int x = 10;
            int y = 5;
            int result;

            Console.WriteLine("x + y = {0}", x + y);
            Console.WriteLine("x - y = {0}", x - y);

            Console.WriteLine("x * y = {0}", x * y);
            Console.WriteLine("x / y = {0}", x / y);
        }
    }
}
```

Çıktı

```
x + y =  15
x - y = 5
x * y = 50
x / y = 2
```

Burada iki noktaya dikkat ediniz.

1. string tipi çıktıların içine parametre değerleri { } Yer Tutucu operatörü ile yerleştirilmektedir. Bu operatör istenen parametreyi çıktıda istenilen yere yerleştir.
2. 10 / 5 bölme işleminin sonucu tamsayı olduğu için, işlemin sonucu doğru olarak çıkmıştır. Ancak, bölüm tamsayı olmadığında çıktı, doğru sonuç yerine bölümün tamsayı kısmını verecektir. Bunu aşağıdaki örnekte inceleyelim.

Operator02.cs

```
using System;
namespace Operatörler
{
  class TamsayıBölme
  {
    static void Main(string[] args)
    {
      int m = 10;
```

```
      int n = 4;
      int    p = m / n;
      float  x = m / n;
      double y = m / n;
      Console.WriteLine("p = {0} ", p);
      Console.WriteLine("x = {0} ", x);
      Console.WriteLine("y = {0} ", y);
    }
  }
}
```

Çıktı

```
p = 2
x = 2
y = 2
```

Bu programın çıktısının neden böyle olduğunu anlamak kolaydır. Programdaki m=10 tamsayısı n=4 tamsayısına tam bölünemez. Gerçek bölüm 2.5 dir. Ancak m/n = 10/4 tamsayı bölme işlemi, bölümün kesirli kısmını atıp, yalnızca tamsayı kısmını almaktadır. O nedenle,

int p	bellekteki adresine 2 yazılır.
float x	bellekteki adresine 2.0000000 yazılır.
Double y	bellekteki adresine 2.000000000000000 yazılır.

Aşağıdaki program, tamsayı bölme işleminden çıkan bölümü eksiksiz yazdırmanın yöntemlerini vermektedir. Satır satır inceleyiniz ve her satırın ne iş yaptığını algılayınız.

Operator03.cs

```
using System;

class Bölme
{
    public static void Main()
    {
        int x, y, sonuç;
        float fsonuç;

        x = 7;
        y = 5;

        sonuç = x / y;
```

```
            Console.WriteLine("x/y: {0}", sonuç);

            fsonuç = (float)x / y;
            Console.WriteLine("x/y: {0}", fsonuç);

            fsonuç = x / (float)y;
            Console.WriteLine("x/y: {0}", fsonuç);

            fsonuç = (float)x / (float)y;
            Console.WriteLine("x/y: {0}", fsonuç);

            fsonuç = (float)(x / y);
            Console.WriteLine("x/y: {0}", fsonuç);
        }
    }
```

Çıktı

```
x/y: 1
x/y: 1,4
x/y: 1,4
x/y: 1,4
x/y: 1
```

Birinci çıktı : sonuç = x / y = 7/5 bir tamsayı bölme işlemi olduğundan bölümün tamsayı kısmı alınmıştır.

İkinci çıktı : fsonuç = (float)x / y = (float)7 / 5 deyiminde, önce 7 sayısı float tipine dönüştürülüyor, sonra 5 sayısına bölünüyor. Bir float tipin bir tamsayıya bölümü gene float tipindendir. Dolayısıyla, (float)7 / 5 = 1.4 dür.

Üçüncü çıktı : Bu çıktı ikinci çıktının simetriğidir. fsonuç = x / (float)y = 7 / (float)5 deyiminde, önce 5 sayısı float tipine dönüştürülüyor, sonra 7 tamsayısı 5.0 sayısına bölünüyor. Bir tamsayı tipin bir float tipine bölümü gene float tipindendir. Dolayısıyla, 7 / (float)5 = 1.4 dür.

Dördüncü çıktı : İkinci ve üçüncü çıktının birleşidir. fsonuç = (float)x / (float)y = (float)7 / (float)5 deyiminde, önce 7 ve 5 sayılarının her ikisi de float tipine dönüşür. Sonra iki float tipin birbirine bölümü yapılır. Bu işlemin sonucu, doğal olarak bir float tipidir. Dolayısıyla, (float)7 / (float)5 = 1.4 dür.

Beşinci çıktı : fsonuç = (float)(x / y) = (float)(7/5) deyiminde, önce (7 / 5) bölme işlemi yapılır. Bu bir tamsayı bölme işlemi olduğu için, birinci çıktıda olduğu gibi, çıkan sonuç 1 dir. (float)1 = 1.0000000 olduğundan, çıktı 1 dir.

Operator04.cs

```
using System;


namespace Operatörler
{
  class Modulo
```

```
  {
    static void Main(string[] args)
    {
     double x = 13;
     double y = 7;
     double sonuç = x % y;
     Console.WriteLine("x % y = {0}", sonuç);
    }
  }
}
```

Çıktı

```
x % y = 6
```

`x % y modulo` işlemi, x sayısının y sayısına bölümünden kalanı verir. 13 sayısı 7 sayısına bölünürse kalan 6 dır.

`Modula` operatörü kesirli sayılar için de tanımlıdır. Bunu aşağıdaki örnekten görebiliriz.

Operator05.cs

```
using System;

namespace Operatörler
{
  class KesirliModulo
  {
    static void Main(string[] args)
    {
     double x = 10.57;
     double y = 4.83;
     double sonuç = x % y;
     Console.WriteLine("x % y = {0}", sonuç);
    }
  }
}
```

Çıktı

```
x % y = 0.91
```

x % y modulo işlemi, x sayısının y sayısına bölümünden kalanı verir. 10 sayısı 4 sayısına bölünürse kalan 2 dir. `Modula` operatörü kesirli sayılar için de tanımlıdır. Bunu aşağıdaki örnekten görebiliriz.

Operator06.cs

```
using System;

namespace Operatörler
```

```
{
    class Aritmetik
    {
        // Aritmetik operatörlerin kullanılışı

        static void Main()
        {
            int toplam=0, fark=0, çarpım=0, modulo=0;
            float bölüm=0;                  // bölümün sonucu
            int sayı1 = 10, sayı2 = 2;      // işleme giren sayılar
            toplam = sayı1 + sayı2;
            fark   = sayı1 - sayı2;
            çarpım = sayı1 * sayı2;
            bölüm  = sayı1 / sayı2;
            modulo = 3 % sayı2;             // 3/2 nin kalanı

            Console.WriteLine("sayı1 = {0}, sayı2 = {1}", sayı1, sayı2);
            Console.WriteLine();
            Console.WriteLine("{0} + {1} = {2}", sayı1, sayı2, toplam);
            Console.WriteLine("{0} - {1} = {2}", sayı1, sayı2, fark);
            Console.WriteLine("{0} x {1} = {2}", sayı1, sayı2, çarpım);
            Console.WriteLine("{0} / {1} = {2}", sayı1, sayı2, bölüm);
            Console.WriteLine();
          Console.WriteLine("3 sayısı {0} ile bölününce  {1}  kalır ",
sayı2, modulo);
        }
    }
}
```

Çıktı

sayı1 = 10, sayı2 = 2

10 + 2 = 12
10 - 2 = 8
10 x 2 = 20
10 / 2 = 5

3 sayısı 2 ile bölününce 1 kalır

İpucu

Bu program çok basittir ve ayrı bir açıklamaya gerek yoktur. Ancak, çıktıya değişken değerini yerleştirme yöntemine tekrar dikkat çekmekte yarar görüyoruz.

```
Console.WriteLine("{0} + {1} = {2}", sayı1, sayı2, toplam);
```

deyiminde {0} {1} ve {2} simgeleri, sırasıyla, sayı1, sayı2, toplam parametre değerlerinin "..... " stringi içindeki yerlerini belirlerler.

++ ve -- Operatörleri

Sayısal değişkenlerin değerlerine 1 eklemek ya da 1 çıkarmak için ++ ve -- operatörlerini kullanırız. x sayısal bir değişken ise

```
x++ = x+1        x-- = x-1
++x = x+1        --x = x-1
```

eşitlikleri vardır. Ancak bu iki operatör, bir sayıya 1 eklemek ya da çıkarmaktan daha fazlasını yapar. Takının değişkenin önüne ya da sonuna gelmesi, işlem sonuçlarını bütünüyle etkiler. Bunu aşağıdaki örneklerden daha kolay anlayacağız.

Önel (Prefix) ve Sonal (Postfix) Takılar

```
int    x = 5;
float  y = 15.63 ;
```

bildirimleri yapılmış olsun.

x++ = 6	x-- = 4	++x = 6	--x = 4
y++ = 16.63	y-- = 14.63	++y = 16.63	-- y= 14.63

olur. Ancak, x ve y işleme giriyorsa, önel ve sonal operatörler farklı etkiler yaratır.

Önel operatör alan değişken değeri işleme girmeden önce değişir, işleme sonra girer.

Sonal operatör alan değişken değeri önce işleme girer, sonra değeri değişir.

Bunu şu işlemlerle daha kolay açıklayabiliriz.

x=5 iken --x * --x = 12	// [4*3 = 12]	ve	x = 3 olur.
x=5 iken ++x * ++x = 42	// [6 * 7 = 42]	ve	x = 7 olur.
x=5 iken x-- * x-- = 20	// [5 * 4 = 20]	ve	x = 4 olur.
x=5 iken x++ * x++ = 30	// [5 * 6 = 30]	ve	x = 7 olur.

Şimdi bu operatörlerin işlevlerini program içinde görelim.

Operator07.cs

```
using System;
namespace Operatörler
{
    class Artım
    {
        static void Main(string[] args)
        {
            int x = 5;
            Console.WriteLine("x = {0}", x);
            Console.WriteLine("++x değeri = {0}", ++x);
            Console.WriteLine("x = {0}", x);
            x = 5;
            Console.WriteLine("x++ değeri = {0}", x++);
            Console.WriteLine("x = {0}", x);
        }
    }
}
```

Çıktı

```
x = 5
++x değeri = 6
x = 6
x++ değeri = 5
x = 6
```

Operator08.cs

```
using System;
class ÖnelSonalOperatörler
{
    public static void Main()
    {
        int n = 35;
        float x = 12.7f;

        Console.WriteLine("n = {0} iken --n = {1} ve n= {2} olur. ", n,
--n, n);
        Console.WriteLine("n = {0} iken ++n = {1} ve n= {2} olur. ", n, +
+n, n);
        Console.WriteLine("n = {0} iken n-- = {1} ve n= {2} olur. ", n,
n--, n);
        Console.WriteLine("n = {0} iken n++ = {1} ve n= {2} olur. ", n,
n++, n);

        Console.WriteLine();

        Console.WriteLine("x = {0} iken --x = {1} ve x= {2} olur. ", x,
--x, x);
        Console.WriteLine("x = {0} iken ++x = {1} ve x= {2} olur. ", x, +
+x, x);
        Console.WriteLine("x = {0} iken x-- = {1} ve x= {2} olur. ", x,
x--, x);
        Console.WriteLine("x = {0} iken x++ = {1} ve x= {2} olur. ", x,
x++, x);


    }
}
```

Çıktı

n = 35 iken --n = 34 ve n= 34 olur.
n = 34 iken ++n = 35 ve n= 35 olur.
n = 35 iken n-- = 35 ve n= 34 olur.
n = 34 iken n++ = 34 ve n= 35 olur.

x = 2,7 iken --x = 1,7 ve x= 1,7 olur.
x = 1,7 iken ++x = 2,7 ve x= 2,7 olur.
x = 2,7 iken x-- = 2,7 ve x= 1,7 olur.
x = 1,7 iken x++ = 1,7 ve x= 2,7 olur.

Operator09.cs

```
using System;

class Unary
{
    public static void Main()
    {
        int sayı = 0;
        int önelArtım;
        int önelEksim;
```

```
        int sonalArtım;
        int sonalEksim;
        int pozitif;
        int negatif;
        sbyte bitNot;
        bool logNot;

        önelArtım = ++sayı;
        Console.WriteLine("önel-Artım: {0}", önelArtım);

        önelEksim = --sayı;
        Console.WriteLine("önel-Eksim: {0}", önelEksim);

        sonalEksim = sayı--;
        Console.WriteLine("Sonal-Eksim: {0}", sonalEksim);

        sonalArtım = sayı++;
        Console.WriteLine("Sonal-Artım: {0}", sonalArtım);

        Console.WriteLine("sayı 'nin son değeri: {0}", sayı);

        pozitif = -sonalArtım;
        Console.WriteLine("Pozitif: {0}", pozitif);

        negatif = +sonalArtım;
        Console.WriteLine("Negatif: {0}", negatif);

        bitNot = 0;
        bitNot = (sbyte)(~bitNot);
        Console.WriteLine("Bitwise Not: {0}", bitNot);

        logNot = false;
        logNot = !logNot;
        Console.WriteLine("Logical Not: {0}", logNot);
    }
}
```

Çıktı

```
Önel-Artım: 1
Önel-Eksim: 0
Sonal-Eksim: 0
Sonal-Artım: -1
sayı 'nın son değeri: 0
Pozitif: 1
Negatif: -1
Bitwise Not: -1
Logical Not: True
```

Operator10.cs

```
using System;
namespace Methods
{
    class ArtımEksim
    {
        public static void Main()
```

```
        {
            int x = 5;
            Console.WriteLine("x={0}ise x-- * x-- = {1}", x, x-- * x--);
            Console.WriteLine("x={0}ise x++ * x++ = {1}", x, x++ * x++);
            Console.WriteLine("x={0}ise x-- * x++ = {1}", x, x-- * x++);
            Console.WriteLine("x={0}ise --x * --x = {1}", x, --x * --x);
            Console.WriteLine("x={0}ise --x * x   = {1}", x, x-- * x);
            Console.WriteLine("x={0}ise --x * ++x = {1}", x, --x * ++x);
            Console.WriteLine("x={0}ise x * x--   = {1}", x, x * x--);
            Console.WriteLine("x={0}ise x-- * ++x = {1}", x, x-- * ++x);
        }
    }
}
```

Çıktı

```
x=5ise x-- * x-- = 20
x=3ise x++ * x++ = 12
x=5ise x-- * x++ = 20
x=5ise --x * --x = 12
x=3ise --x * x   = 6
x=2ise --x * ++x = 2
x=2ise x * x--   = 4
x=1ise x-- * ++x = 1
```

Operator11.cs

```
using System;

namespace Operatörler
{
    class İkiliİşlem
    {
        public static void Main()
        {
            int x, y, sonuç;
            float fSonuç;

            x = 7;
            y = 5;

            sonuç = x + y;
            Console.WriteLine("{0} + {1} = {2}", x, y, sonuç);

            sonuç = x - y;
            Console.WriteLine("{0} - {1} = {2}", x, y, sonuç);

            sonuç = x * y;
            Console.WriteLine("{0} * {1} = {2}", x, y, sonuç);

            sonuç = x / y;
            Console.WriteLine("{0} / {1} = {2}", x, y, sonuç);

            fSonuç = (float)x / (float)y;
            Console.WriteLine("{0} / {1} = {2}", x, y, fSonuç);
```

```
            sonuç = x % y;
            Console.WriteLine("{0} % {1} = {2}", x, y, sonuç);
        }
    }
}
```

Çıktı

```
7 + 5 = 12
7 - 5 = 2
7 * 5 = 35
7 / 5 = 1
7 / 5 = 1,4
7 % 5 = 2
```

Atama Operatörleri

Atama operatörleri, değişkenlere değer atamak için kullanılan simgelerdir. C# dilinde aşağıdaki atama operatörleri kullanılır:

Operatör	Açıklama
=	Sağdaki değeri soldaki değişkene atar.
+=	Soldakine sağdakini ekler, sonucu soldakine atar.
-=	Soldakinden sağdakini çıkarır, sonucu soldakine atar.
*=	Soldakini sağdaki ile çarpar, sonucu soldakine atar.
/=	Soldakini sağdakine böler, sonucu soldakine atar.
%=	Soldaki ile sağdakinin modula işleminin sonucunu soldakine atar.

Operator12.cs

```
using System;
namespace Operatörler
{
    class Atama1
    {
        static void Main(string[] args)
        {
            int x = 5;
            int y = 5;
            Console.WriteLine("x = {0} ve y = {1}", x ,  y);
            x = x + y;
            Console.WriteLine("x = x + y ise x = {0}", x + y);
            x += y;
            Console.WriteLine("x += y ise x = {0}", x);

            Console.WriteLine("**********");
            x = x - y;
            Console.WriteLine("x = x - y ise x = {0}", x - y);
            x -= y;
            Console.WriteLine("x -= y  ise x = {0}", x);
            Console.WriteLine("**********");
            x = x * y;
            Console.WriteLine("x = x * y ise  x = {0}", x * y);
            x *= y;
            Console.WriteLine("x *= y ise x = {0}", x);
```

```
            Console.WriteLine("**********");
            x = x / y;
            Console.WriteLine("x = x / y and x = {0}", x / y);
            x /= y;
            Console.WriteLine("x /= y  ise x = {0}", x);
            Console.WriteLine("**********");
            x = x % y;
            Console.WriteLine("x = x % y is x = {0}", x % y);
            x %= y;
            Console.WriteLine("x %= y  ise  x = {0}", x);
        }
    }
}
```

Çıktı

```
x = 5 ve y = 5
x = x + y ise x = 15
x += y ise x = 15
**********
x = x - y ise x = 5
x -= y  ise x = 5
**********
x = x * y ise  x = 125
x *= y ise x = 125
**********
x = x / y and x = 5
x /= y  ise x = 5
**********
x = x % y is x = 0
x %= y  ise  x = 0
```

Operator13.cs

```
using System;

namespace Methods
{
    class ArtımEksim
    {
        public static void Main()
        {
            int x = 5;
            int y = 4;
            Console.WriteLine("x={0} ve y = {1} ise x +=y = {2}", x, y ,
x += y);
            Console.WriteLine("x={0} ve y = {1} ise x -=y = {2}", x, y ,
x -= y);
            Console.WriteLine("x={0} ve y = {1} ise x *=y = {2}", x, y ,
x *= y);
            Console.WriteLine("x={0} ve y = {1} ise x /=y = {2}", x, y ,
x /= y);
            Console.WriteLine("x={0} ve y = {1} ise x %=y = {2}", x, y ,
x %= y);
        }
```

```
        }
    }
```

Çıktı

```
x=5   ve y = 4 ise      x+=y  = 9
x=9   ve y = 4 ise      x-=y  = 5
x=5   ve y = 4 ise      x*=y  = 20
x=20  ve y = 4 ise      x/=y  = 5
x=5   ve y = 4 ise      x%=y  = 1
```

İlişkisel Operatörler

İlişkisel (relational) operatörler iki değişkenin değerlerini karşılaştırır. Dolayısıyla, karşılaştırılan öğeler arasında bir boolean deyim kurar. Karşılaştırılan değerlerin eşitliğini ya da birinin ötekinden büyük ya da küçük olduğunu belirten boolean deyim doğru ya da yanlış (true, false) değerlerden birisini alır. C# dilinde aşağıdaki ilişkisel operatörler kullanılır.

Operatör	Açıklama
==	Eşit mi?
!=	Eşit-değil mi?
>	Büyük mü?
<	Küçük mü?
>=	Büyük veya eşit mi?
<=	Küçük veya eşit mi?

İlişkisel operatörler, bir boolean deyim içinde yer alırlar ve daima true veya false değerlerinden yalnızca birisini alırlar. Örneğin,

```
int sayı1 = 7, sayı2 = 8;
```

ise,

```
sayı1 == sayı2  // false
sayı1 != sayı2  // true
sayı1 >  sayı2  // false
sayı1 <  sayı2  // true
sayı1 <= sayı2  // true
sayı1 >= sayı2  // false
```

olur.

Uyarı

İlişkisel operatörler, yalnızca birbirleriyle mukayese edilebilir veriler arasında ilişki kurar. Örneğin, iki sayı mukayese edilebilir, eşit olup olmadıkları ya da birinin ötekinden büyük olup olmadığını anlamak için ikisi arasında bir ilişkisel operatör kurulabilir. Ama, bir metin ile bir sayıyı ya da bir sayı ile bir mantıksal değeri mukayese edemeyiz.

Örnek

```
int     sayı = 5;
bool    mnt  = true;
```

bildirimi yapılmış olsun. Sayı ile mnt mantıksal değeri arasında, yukarıda tanımlanan altı ilişkisel bağıntıdan hiç birisi kurulamaz. Dolayısıyla, sayı == mnt ya da sayı < mnt gibi deyimler derleyici tarafından derlenemez.

Operator13.cs

```
using System;
namespace Operators
{
    class İlişkiselOperatörler
    {
        static void Main(string[] args)
        {
            int x = 10;
            int y = 4;
            bool sonuç;

            sonuç = (x > y);
            Console.WriteLine("x > y = {0}", sonuç);
            sonuç = (x < y);
            Console.WriteLine("x < y = {0}", sonuç);
            sonuç = (x <= y);
            Console.WriteLine("x <= y = {0}", sonuç);
            sonuç = (x >= y);
            Console.WriteLine("x >= y = {0}", sonuç);
            sonuç = (x == y);
            Console.WriteLine("x == y = {0}", sonuç);
            sonuç = (x != y);
            Console.WriteLine("x != y = {0}", sonuç);
        }
    }
}
```

Çıktı

```
x > y     = True
x < y     = False
x <= y    = False
x >= y    = True
x == y    = False
x != y    = True
```

Mantıksal (Logic) Operatörler

C# dilinde, mantıksal deyimleri birbirlerine bağlamak için iki operatör kullanılır

```
&&    Logical  AND      (Mantıksal VE)
||    Logical  OR       (Mantıksal VEYA)
```

&& Operatörü

boolean1 ve boolean2 iki mantıksal deyim olmak üzere, bu iki deyimin mantıksal VE ile birbirlerine bağlanması için

```
boolean1 && boolean2
```

yazılır. Bu yeni bir mantıksal deyimdir. Ancak bileşenlerinden her ikisi de doğru olduğunda bileşke deyim doğru, aksi halde yanlıştır.

boolean1 yanlış ise, boolean2 nin değeri hesaplanmaz. Çünkü boolean2 doğru olsa bile bileşke deyim yanlış olacaktır. O nedenle, bu deyime bazen *kısa-devre-VE* mantıksal deyimi denir.

Aşağıdaki programın yalnızca ilk deyimi denetlediğini, ikinci deyimi denetlemeye gerek kalmadığını görebilirsiniz.

Operator14.cs

```
using System;
class Test
{
    static bool aaaa()
    {
        Console.WriteLine("aaaa fonksiyonu çağrıldı");
        return false;
    }

    static bool bbbb()
    {
        Console.WriteLine("bbbb fonksiyonu çağrıldı");
        return true;
    }

    public static void Main()
    {
        Console.WriteLine("regular AND:");
        Console.WriteLine("Sonuç : {0}", aaaa() & bbbb());
        Console.WriteLine("kısa-devre AND:");
        Console.WriteLine("Sonuç : {0}", aaaa() && bbbb());
    }
}
```

Çıktı

```
regular AND:
aaaa fonksiyonu çağrıldı
bbbb fonksiyonu çağrıldı
Sonuç : False
kısa-devre AND:
aaaa fonksiyonu çağrıldı
Sonuç : False
```

Operator15.cs

```
using System;
using System.Globalization;
using System.Threading;

namespace Methods
{
    class ArtımEksim
    {
        public static void Main()
        {
            int x = 5;
            int y = 4;

            Console.WriteLine(5 == 6-1 && 7 > 6  );
            Console.WriteLine(5 >= 4 && 7 < 6 + 3);
            Console.WriteLine(5 != 4 && 7-1 == 6);
```

```
            Console.WriteLine(!(5 == 4) && 7 > 6);
        }
    }
}
```

Çıktı

```
True
True
True
True
```

|| Operatörü

`boolean1` ve `bolean2` iki mantıksal deyim olmak üzere, bu iki deyimin mantıksal `VEYA` ile birbirlerine bağlanması için

```
boolean1 || boolean2
```

yazılır. Bu yeni bir mantıksal deyimdir. Bileşenlerinden herhangi birisi doğru olduğunda bileşke deyim doğru, ancak her iki bileşeni yanlış olduğunda bileşke deyim yanlış olur.

boolean1 doğru ise, boolean2 nin değeri hesaplanmaz. Çünkü boolean2 yanlış olsa bile bileşke deyim doğru olacaktır. O nedenle, bu deyime bazen kısa-devre_VEYA mantıksal deyimi denir.

Aşağıdaki programın yalnızca ilk deyimi denetlediğini, ikinci deyimi denetlemeye gerek kalmadığını görebilirsiniz.

Operator16.cs

```
using System;

namespace Methods
{
    class ArtımEksim
    {
        public static void Main()
        {
            int x = 5;
            int y = 4;

            Console.WriteLine(5 < 6-1 || 7 > 6  );
            Console.WriteLine(5 >= 4 || 7 < 6 + 3);
            Console.WriteLine(5 != 4 || 7 - 1 == 6);
            Console.WriteLine(!(5 == 4) || 7 > 6);
        }
    }
}
```

Çıktı

```
True
True
True
True
```

Operator17.cs

```
using System;
class Test
{
  static bool cccc()
  {
   Console.WriteLine("cccc metodu çağrıldı");
    return true;
  }

  static bool dddd()
  {
   Console.WriteLine("dddd metodu çağrıldı");
    return false;
  }

  public static void Main()
  {
    Console.WriteLine("regular OR:");
    Console.WriteLine("sonuç : {0}", cccc() | dddd());
    Console.WriteLine("kısa-devre OR:");
    Console.WriteLine("sonuç : {0}", cccc() || dddd());
  }
}
```

Çıktı

```
regular OR:
cccc metodu çağrıldı
dddd metodu çağrıldı
sonuç : True
kısa-devre OR:
cccc metodu çağrıldı
sonuç :  True
```

Bitsel (Bitwise) Operatörler

Bitsel işlemler yapmak için kullanılan operatörledir.

Operatör	Açıklama
&	bitsel AND
|	bitsel OR
^	bitsel XOR

!	bitsel NOT

Bunlardan ilk üçü, programda çok gerekli ise kullanılır. Sonuncu ile daha çok karşılaşabiliriz. && ve || mantıksal operatörlerinin farklı kullanım biçimlerini ileride örnekler üzerinde göreceğiz. Aşağıdakiler bu konuda biraz ipucu verebilirler.

```
bool    aa     = false;
bool    bb    = !aa;
```

ise bb nin değeri true olur.

```
     int  i=8, j=14;
     bool xxx  = i>4 && j<12;
```

ise, xxx false değerini alır. Oysa,

```
     bool yyy = i>3 || j<10;
```

ise yyy 'nin değeri true olur.

```
     bool zzz    =    (i>3 && j<10) || (i<9 && j>10)
```

ise, zzz nin değeri true olur.

Başka Operatörler

C# dilinde aşağıdaki operatörleri de kullanırız.

Operand	Açıklama	
<<	Sola kayan bit işlemi	(left shift bitwise operator)
>>	Sağa kayan bit işlemi	(right shift bitwise operator)
.	Nesne öğelerine erişim	(member access for objects)
[]	Array indisleme	(indexing operator used in arrays and collections)
()	Veri dönüştürme operatörü	(cast operator)
?:	Koşullu deyim operatörü	(ternary operator)

Operatör Öncelikleri

Bir deyimde birden çok operatör kullandığımızda hangi operatörün hangisinden önce işlevini yapacağını bilmeliyiz. Bunu basit bir örnekle açıklayalım: a ile b sayılarını toplayıp, toplamı 2 ile çarpmak isteyelim.

```
a+b * 2
```

formülü yanlış olacaktır. Çünkü, C# dilinde * operatörü + operatöründen önce işleme girer. Örneğin,

```
     int sayı  =  12+3 * 2 ;
```

deyiminin sonucu, beklentimiz olan 30 değil 18 dir. Çünkü, derleyicimiz 12+3 * 2 işlemini şu sırada yapacaktır:

```
3 * 2 + 12 = 18.
```

C# operatörlerinin öncelik sırası
Üst öncelikten alt önceliğe doğru sıralıdır

Operator Kategorisi	Operatörler
Primary	x.y f(x) a[x] x++ x--
Unary	+ - ! ~ ++x --x (T)x
Çarpımsal(Multiplicative)	* / %
Toplamsal (Additive)	+ -
Kayma (Shift)	<< >>
İlişkisel (Relational)	< > <= >= is as
Eşitlik (Equality)	== !=
Mantıksal VE (Logical AND)	&
Mantıksal XOR (Logical XOR)	^
Mantıksal VEYA (Logical OR)	|
Koşullu VE (Conditional AND)	&&
Koşullu VEYA (Conditional OR)	||
Koşullu (Conditional, ternary)	?:
Atama (Assignment)	= *= /= %= += -= <<= >>= &= ^= |=

Alıştırmalar

Aşağıdaki program unary operatörlerin kullanılışını göstermektedir. Programı satır satır çıktı ile karşılaştırarak çözümleyiniz.

Operatör18.cs

```
using System;

class Unary
{
    public static void Main()
    {
        int unary = 0;
        int önelArtım;
        int önelEksim;
        int sonalArtım;
        int sonalEksim;
        int artıSayı;
        int eksiSayı;
        sbyte bitselNOT;
        bool mantıksalNOT;
```

```
        önelArtım = ++unary;
        Console.WriteLine("Önel Artm  : {0}", önelArtım);

        önelEksim = --unary;
        Console.WriteLine("Önel Eksim : {0}", önelEksim);

        sonalEksim = unary--;
        Console.WriteLine("Sonal Artım: {0}", sonalEksim);

        sonalArtım = unary++;
        Console.WriteLine("Sonal Eksim: {0}", sonalArtım);

        Console.WriteLine("Unary: {0}", unary);

        artıSayı = -sonalArtım;
        Console.WriteLine("Artı : {0}", artıSayı);

        eksiSayı = +sonalArtım;
        Console.WriteLine("Eksi : {0}", eksiSayı);

        bitselNOT = 0;
        bitselNOT = (sbyte)(~bitselNOT);
        Console.WriteLine("Bitwise Not  : {0}", bitselNOT);

        mantıksalNOT = false;
        mantıksalNOT = !mantıksalNOT;
        Console.WriteLine("Mantıksal Not: {0}", mantıksalNOT);
    }
}
```

Çıktı

```
Önel Artm  : 1
Önel Eksim : 0
Sonal Artım : 0
Sonal Eksim : -1
Unary: 0
Artı  : 1
Eksi  : -1
Bitwise Not  : -1
Mantıksal Not: True
```

OPERATÖRLER
Alıştırmalar

Aritmetik Operatörleri

Operatör	*Açıklama*	*Grup*
+	Toplama, artı işleci	İkili İşlem (binary operator)
-	Çıkarma, eksi işleci	İkili İşlem (binary operator)
*	Çarpma, çarpım işleci	İkili İşlem (binary operator)
/	Bölme, bölüm işleci	İkili İşlem (binary operator)
%	Modulo, kalan işleci	İkili İşlem (binary operator)
++	Artım (auto increment)	Birli İşlem (unary operator)
--	Eksim (auto decrement)	Birli İşlem (unary operator)

Örnek:

```
using System;

class Binary
{
    public static void Main()
    {
        int x, y, sonuç;
        float floatsonuç;

        x = 7;
        y = 5;

        sonuç = x + y;
        Console.WriteLine("x+y: {0}", sonuç);

        sonuç = x - y;
        Console.WriteLine("x-y: {0}", sonuç);

        sonuç = x * y;
        Console.WriteLine("x*y: {0}", sonuç);

        sonuç = x / y;
        Console.WriteLine("x/y: {0}", sonuç);

        floatsonuç = (float)x / (float)y;
```

```
        Console.WriteLine("x/y: {0}", floatsonuç);

        sonuç = x % y;
        Console.WriteLine("x%y: {0}", sonuç);

        sonuç += x;
        Console.WriteLine("sonuç+=x: {0}", sonuç);
    }
}
```

Çıktı:

```
x+y: 12
x-y: 2
x*y: 35
x/y: 1
x/y: 1.4
x%y: 2
sonuç+=x: 9
```

Örnek:

```
using System;
namespace Operator
{
    class Aritmetik
    {
        static void Main(string[] args)
        {
            int x = 10;
            int y = 5;
            int sonuç;

            sonuç = x + y;
            Console.WriteLine("x + y = {0}", sonuç);

            sonuç = x - y;
            Console.WriteLine("x - y = {0}", sonuç);

            sonuç = x * y;
            Console.WriteLine("x * y = {0}", sonuç);

            sonuç = x / y;
            Console.WriteLine("x / y = {0}", sonuç);
        }
    }
}
```

Çıktı:

```
x + y = 15
x - y = 5
x * y = 50
x / y = 2
```

Yukarıdaki işlemleri **sonuç** değişkenini hiç kullanmadan yapabiliriz:

Örnek:

```
using System;
namespace Operator
{
    class Aritmetik
    {
        static void Main(string[] args)
        {
            int x = 10;
            int y = 5;
            int sonuç;
            Console.WriteLine("x + y = {0}", x + y);

            Console.WriteLine("x - y = {0}", x - y);

            Console.WriteLine("x * y = {0}", x * y);

            Console.WriteLine("x / y = {0}", x / y);
        }
    }
}
```

Örnek:

```
using System;
class Modulo
{
    static void Main()
    {

        int a = 2;
        int b = 6;

        int c = 12;
        int d = 5;

        Console.WriteLine(b % a);
        Console.WriteLine(c % d);
        Console.Read();
    }
}
```

Çıktı

```
0
2
```

Örnek:

```
using System;

namespace operators
{
    class Aritmetik03
    {
        static void Main(string[] args)
        {
            double x = 10.34;
            double y = 4;
            double result = x % y;
            Console.WriteLine("x % y = {0}", result);

        }
    }
}
```

Çıktı:

```
x % y = 2,34
```

Tamsayı Bölme

/ bölme işlemi, aksi istenmediği zaman bölümün tamsayı kısmını verir. Eğer bölümün kesir kısmını istiyorsak, bunu istemli dönüşüm (explicit casting) ile belirtmeliyiz.

Örnek:

```
using System;

class Binary
{
    public static void Main()
    {
        int x, y, sonuç;
        float floatsonuç;

        x = 7;
        y = 5;

        sonuç = x + y;
```

```
        Console.WriteLine("x+y: {0}", sonuç);

        sonuç = x - y;
        Console.WriteLine("x-y: {0}", sonuç);

        sonuç = x * y;
        Console.WriteLine("x*y: {0}", sonuç);

        sonuç = x / y;
        Console.WriteLine("x/y: {0}", sonuç);

        floatsonuç = (float)x / (float)y;
        Console.WriteLine("x/y: {0}", floatsonuç);

        sonuç = x % y;
        Console.WriteLine("x%y: {0}", sonuç);

        sonuç += x;
        Console.WriteLine("sonuç+=x: {0}", sonuç);
    }
}
```

Çıktı:

```
x+y: 12
x-y: 2
x*y: 35
x/y: 1
x/y: 1,4
x%y: 2
sonuç+=x: 9
Devam etmek için bir tuşa basın . . .
```

int, float ve double tiplerinin işleme girişleri:

Örnek:

```
using System;
class Test
{
    static void Main()
    {

        int a = 2;
        int b = 6;
        int c = a - b;

        float d = 1;
        float e = 11.1f;
        double f = 1;

        Console.WriteLine(c);
```

```
        Console.WriteLine(d + e);
        Console.WriteLine(f + e);
    }
```

Çıktı

```
-4
12,1
12,1000003814697
```

Önel ve Sonal Artım ve Eksim (++, --)

Örnek:

```
class Program
{
    static void Main(string[] args)
    {

        int y;
        int a = 100;

        y = ++a;   //önel-artım (pre-increment)
        Console.WriteLine("y = " + y + " and a = " + a);
        //y=101  and a=101

        y = a++;   //sonal-artım (post-increment)
        Console.WriteLine("y = " + y + " and a = " + a);
        //y=101  and a=102
    }
}
```

Çıktı:

```
y = 101 and a = 101
y = 101 and a = 102
```

Örnek:

```
using System;

class Unary
{
    public static void Main()
    {
```

```
        int unary = 0;
        int preIncrement;
        int preDecrement;
        int postIncrement;
        int postDecrement;
        int positive;
        int negative;
        sbyte bitNot;
        bool logNot;

        preIncrement = ++unary;
        Console.WriteLine("pre-Increment: {0}", preIncrement);

        preDecrement = --unary;
        Console.WriteLine("pre-Decrement: {0}", preDecrement);

        postDecrement = unary--;
        Console.WriteLine("Post-Decrement: {0}", postDecrement);

        postIncrement = unary++;
        Console.WriteLine("Post-Increment: {0}", postIncrement);

        Console.WriteLine("Final Value of Unary: {0}", unary);

        positive = -postIncrement;
        Console.WriteLine("Positive: {0}", positive);

        negative = +postIncrement;
        Console.WriteLine("Negative: {0}", negative);

        bitNot = 0;
        bitNot = (sbyte)(~bitNot);
        Console.WriteLine("Bitwise Not: {0}", bitNot);

        logNot = false;
        logNot = !logNot;
        Console.WriteLine("Logical Not: {0}", logNot);
    }
}
```

Çıktı:

```
pre-Increment: 1
pre-Decrement: 0
Post-Decrement: 0
Post-Increment: -1
Final Value of Unary: 0
Positive: 1
Negative: -1
Bitwise Not: -1
Logical Not: True
```

+ ve - operatörleri

+ ve – operatörleri **birli (unary)** operatörlerdir. + operatörü, önüne geldiği sayının işaretini değiştirmezken, - operatörü önüne geldiği sayının işaratini değiştirir; yani artı sayıyı eksi, eksi sayıyı artı yapar. Aşağıdaki örnek bunun nasıl olduğunu göstermektedir.

Örnek:

```
using System;
namespace Operators
{
    class Class1
    {
        static void Main(string[] args)
        {
            int x = -10;
            int y = +10;
            int z = +-10;
            int a = 3;
            int sonuç;
            Console.WriteLine("x = {0}, y = {1}, z = {2}", x, y, z);

             z = -10;
             sonuç = z * a;
            Console.WriteLine("sonuç = {0}", sonuç);
             sonuç = z * +a;
            Console.WriteLine("sonuç = {0}", sonuç);
            Console.ReadLine();
        }
    }
}
```

Çıktı:

```
x = -10, y = 10, z = -10
sonuç = -30
sonuç = -30
```

Atama (assignment) Operatörleri

Atama operatörleri, değişkenlere değer atamak için kullanılan işleçlerdir. C# dilinde aşağıdaki atama operatörleri kullanılır:

Operatör	Açıklama
=	Sağdaki değeri soldaki değişkene atar.
+=	Soldakine sağdakini ekler, sonucu soldakine atar.
-=	Soldakinden sağdakini çıkarır, sonucu soldakine atar.
*=	Soldakini sağdaki ile çarpar, sonucu soldakine atar.
/=	Soldakini sağdakine böler, sonucu soldakine atar.
%=	Soldaki ile sağdakinin modula işleminin sonucunu soldakine atar.

Örnek:

```
using System;
namespace Operators
{
    class Class1
    {
        static void Main(string[] args)
        {
            int x = 8;
            int y = 6;
          Console.WriteLine(" Başlangıçta x = {0} ve  y = {1} dır.", x, y);

          Console.WriteLine("**********");

             x = x + y;
          Console.WriteLine("x = x + y atamasından sonra  x = {0}  olur", x);
             x = 8;  x += y;
          Console.WriteLine("x += y    atamasından sonra  x = {0}  olur", x);

          Console.WriteLine("**********");

             x = 8;  x = x - y;
          Console.WriteLine("x = x - y  atamasından sonra  x = {0}  olur", x);
             x = 8; x -= y;
          Console.WriteLine("x -= y     atamasından sonra  x = {0}  olur", x);

          Console.WriteLine("**********");

             x = 8; x = x * y;
          Console.WriteLine("x = x * y  atamasından sonra  x = {0}  olur", x);
             x = 8; x *= y;
          Console.WriteLine("x *= y     atamasından sonra  x = {0}  olur", x);

          Console.WriteLine("**********");

             x = 8; x = x / y;
```

```
            Console.WriteLine("x = x / y  atamasından sonra  x = {0}  olur", x);
               x = 8; x /= y;
            Console.WriteLine("x /= y     atamasından sonra  x = {0}  olur", x);

            Console.WriteLine("**********");

               x = 8; x = x % y;
            Console.WriteLine("x = x % y  atamasından sonra  x = {0}  olur", x);
               x = 8; x %= y;
            Console.WriteLine("x %= y     atamasından sonra  x = {0}  olur", x);
         }
    }
}
```

Çıktı:

```
Başlangıçta x = 8 ve  y = 6 dır.
**********
x = x + y atamasından sonra  x = 14  olur
x += y    atamasından sonra  x = 14  olur
**********
x = x - y  atamasından sonra  x = 2  olur
x -= y     atamasından sonra  x = 2  olur
**********
x = x * y  atamasından sonra  x = 48  olur
x *= y     atamasından sonra  x = 48  olur
**********
x = x / y  atamasından sonra  x = 1  olur
x /= y     atamasından sonra  x = 1  olur
**********
x = x % y  atamasından sonra  x = 2  olur
x %= y     atamasından sonra  x = 2  olur
```

İlişkisel Operatörler

İlişkisel (relational) operatörler iki değişkenin değerlerini karşılaştırır. Dolayısıyla, karşılaştırılan öğeler arasında bir boolean deyim kurar. Karşılaştırılan değerlerin eşitliğini ya da birinin ötekinden büyük ya da küçük olduğunu belirten boolean deyim `doğru` ya da `yanlış` (`true, false`) değerlerden birisini alır. C# dilinde aşağıdaki ilişkisel operatörler kullanılır.

Operatör	Açıklama
==	Eşit mi?
!=	Eşit-değil mi?
>	Büyük mü?
<	Küçük mü?
>=	Büyük veya eşit mi?
<=	Küçük veya eşit mi?

Mantıksal (Logic) Operatörler

C# dilinde, mantıksal deyimleri birbirlerine bağlamak için iki operatör kullanılır

```
&&   Logical  AND     (Mantıksal VE)
||   Logical  OR      (Mantıksal VEYA)
```

&& Operatörü

boolean1 ve boolean2 iki mantıksal deyim olmak üzere, bu iki deyimin mantıksal VE ile birbirlerine bağlanması için

```
boolean1 && boolean2
```

Bitsel (Bitwise) Operatörler

Bitsel işlemler yapmak için kullanılan operatörledir.

Operatör	Açıklama
&	bitsel AND
|	bitsel OR
^	bitsel XOR
!	**bitsel NOT**

Başka Operatörler

C# dilinde aşağıdaki operatörleri de kullanırız.

Operand	Açıklama	
<<	Sola kayan bit işlemi	(left shift bitwise operator)
>>	Sağa kayan bit işlemi	(right shift bitwise operator)
.	Nesne öğelerine erişim	(member access for objects)
[ ]	Array indisleme	(indexing operator used in arrays and collections)
( )	Veri dönüştürme operatörü	(cast operator)
? :	Koşullu deyim operatörü	(ternary operator)

Bölüm 07

Array

Array Nedir?

Array, aynı tipten çok sayıda değişken tanımlamak için kullanılır. Soyut bir veri yapısıdır. Matematikteki sonlu dizi kavramına benzer. C# dilinde array bir sınıftır. Her sınıfın soyut bir veri tipi olduğunu biliyoruz. Array sınıfı array yaratma, arraylerle işlem yapma, array içinde bileşen arama ve array'in bileşenlerini sıralama gibi array ile ilgili işlemleri yapmaya yarayan öğeleri içeren bir sınıftır.

Array Yaratma

Array bir sınıf olduğuna göre, bir array'i yaratma bir sınıftan nesne yaratma gibidir. Üç aşaması vardır:

```
Birinci Aşama    : Array sınıfının bildirimi
İkinci Aşama     : Array sınıfından array nesnesini yaratma
Üçüncü Aşama     : Array'in bileşenlerine değer atama
```

Şimdi bu üç aşamanın nasıl yapıldığını bir örnek üzerinde görelim.

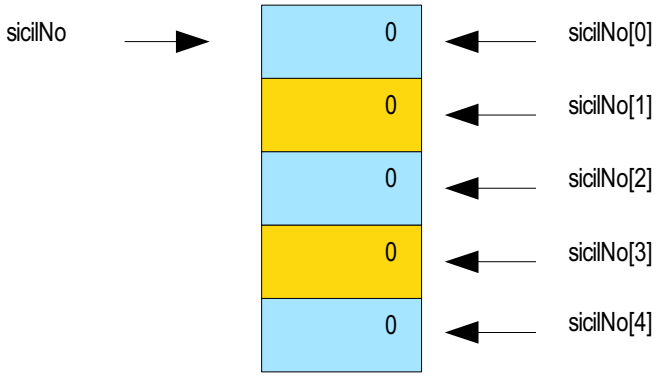
Birinci aşama:

```
int []    sicilNo ;
```

deyimi int tipinden `sicilNo` adlı bir array bildirimidir. Bu aşamada `sicilNo` null işaret eden bir referanstır (işaretçi, pointer).

İkinci aşama:

```
sicilNo = new int[5] ;
```



kurucu (constructor) deyimi `sicilNo` tarafından işaret edilen bir nesne yaratır. Başka bir deyişle, ana bellekte 5 tane `int` değer tutacak bir yer ayırır. `SicilNo` bellekte ayrılan bu yeri işaret eder. O nedenle, `sicilNo`'ya referans (işaretçi, pointer) denmektedir. O halde, arrayler referans tipidir. Başka bir deyişle, array'in kendisi bir değer değil, kendisine ait nesnenin bellekte bulunduğu adresi işaret eden referanstır.

Arrayin işaret ettiği nesnenin içinde `int` tipi veri tutacak 5 tane bellek adresi vardır. Bu adresler

SicilNo[0]

sicilNo[1]

sicilNo[2]

sicilNo[3]

sicilNo[4]

adları tarafından işaret (referans) edilirler.

[] Operatörü

Array adının sonuna gelen `[]` parantezleri, arrayin bileşenlerini, yukarıda gösterildiği gibi, indisleriyle (damga, numara) belirler.

Array'in Bileşenleri

`sicilNo[i]` `(i=0,1,2,3,4)` ler söz konusu nesne içinde birer değişkendir. Bu değişkenler sayesinde, array beş tane int tipi veriyi tutabilme yeteneğine sahip olur. Bu değişkenlere array'in bileşenleri denir. `0,1,2,3,4` sayıları bileşenlerin sıra numaralarıdır; damga (index) adını alırlar. Sıra numaraları (index) daima 0 dan başlar, birer artarak gider. n tane bileşeni olan bir array'in ilk bileşeninin damgası 0, son bileşeninin damgası (n-1) olur. Bir array'in uzunluğu onun bileşenlerinin sayısıdır.

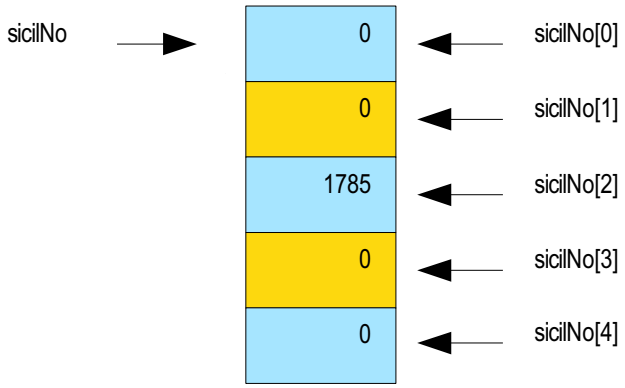
Eğer `new int[5]` yerine new `int[500]` yazsaydık, 5 bileşen yerine 500 bileşen elde ederdik.

Arrayin işaret ettiği nesne yaratılınca, onun bileşenleri kendiliğinden başlangıç değeri alırlar. Bunlara `öndeğer` (default value) denir. Öndeğerler bileşenlerin veri tipine bağlı olarak değişir. C# dilinde bütün sayısal değişkenlerin öndeğerleri daima 0 olur. Referans tiplerde ise `null` olur. O nedenle, yukarıda `sicilNo` referansının işaret ettiği nesne içindeki `SicilNo[0]`, `sicilNo[1]`, `sicilNo[2]`, `sicilNo[3]` , `sicilNo[4]` bileşenlerinin (değişken) öndeğerleri kendiliğinden 0 olur.

Üçüncü aşama:

```
sicilNo[2] = 1785;
```

ataması, array'in üçüncü bileşenine 1785 değerini atar.



İstenirse öteki bileşenlere de benzer yolla atamalar yapılabilir.

Yukarıdaki üç aşamayı birleştirerek iki ya da bir adımda hepsini bitirebiliriz. Örneğin,

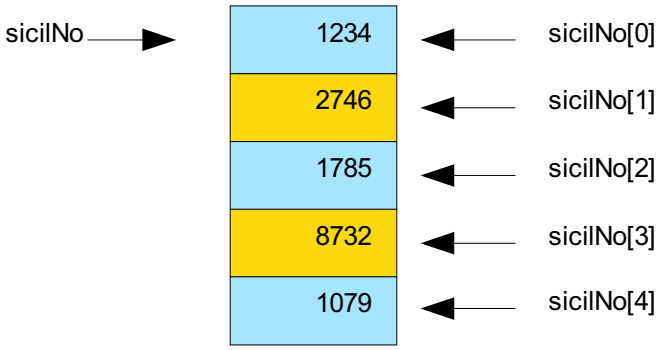
```
int [] sicilNo = new int[] {1234, 2746, 1785, 8732, 1079};
```

deyimi üç aşamayı birden yapar.

İstenirse,

```
int [] sicilNo = new int[5] ;
```

deyimi ile ilk iki aşama tamamlanır, bileşenlere değer atama işi sonraya bırakılabilir.



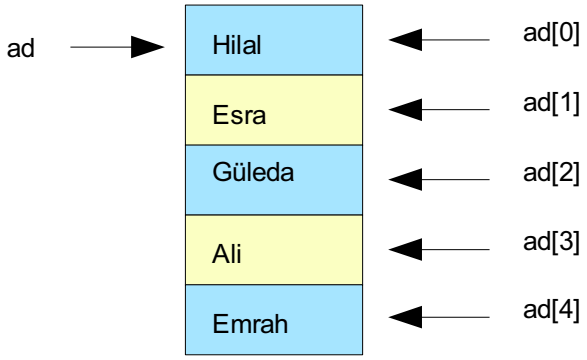
C# dilinde her veri tipinden array yaratılabilir. Örneğin,

```
string [] ad = new string[] {"Hilal", "Esra", "Güleda", "Ali", "Emrah"};
```

deyimi `string` tipinden bir array sınıfı bildirmiş, onun bir nesnesini yaratmış ve o nesnenin bileşenlerine `string` tipi değerler atamıştır. Bu atama

```
ad[0] = "Hilal" ;  ad[1] = "Esra" ;  ad[2] = "Güleda" ;  ad[3] = "Ali" ;
ad[4] = "Emrah" ;
```

atamalarına denktir.



Artık, array yaratmak için genel sözdizimini yazabiliriz:

```
veriTipi []  arrayAdı ;    (array bildirimi)
arrayAdı = new veriTipi [bileşen sayısı];     (array nesnesini yaratma)
```

Array nesnesi yaratıldıktan sonra, bileşenlerine değer atama eylemi, değişkenlere değer atama gibidir. İstenen bileşen indeks sayısı ile seçilerek seçkili (random) değer atanabilir.

Array bildiriminde, yukarıda yaptığımız gibi, arrayin uzunluğunu (bileşenlerinin sayısını) nesneyi yaratırken belirleyebiliriz. Buna sabit uzunluk belirleme diyeceğiz. Ancak, bazı durumlarda, nesneyi yaratırken arrayin uzunluğunu tam bilmiyor olabiliriz. Bu durumda, arrayin uzunluğunu dinamik olarak belirleriz. Dinamik uzunluklu arraye bileşen ekledikçe arrayin uzunluğu kendiliğinden artar. İstendiği an dinamik uzunluklu array sabit uzunluklu array haline getirilebilir. Örneğin,

```
short []  sıraNo ;
```

bildirimi dinamik uzunluklu bir array tanımlar.

```
double []  aylıkÜcret[145];
```

deyimi 145 bileşenli bir array tanımlar.

```
int[] arr;
arr = new int[10];
```

deyimi 10 bileşenli bir array yaratır.

```
int[] arr;
arr = new int[10] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

deyimi 10 bileşenli bir array yaratır ve bileşenlerine değerler atar . Atanan değer sayısı bileşen sayısına eşit olmalıdır. Ayrıca, bileşenlere değerlerin yazılış sırasıyla atandığını unutmayınız. Arrayin belirli bir bileşenine değer atamak isterseniz, Seçkili Erişim paragrafına bakınız.

Array bileşenlerine değer atama eylemi, genellikle program koşarden dinamik olarak yapılır. Aşağıdaki örnek, bir döngü ile array bileşenlerinin bazılarına değer atamaktadır. Sonra bütün bileşenlerin değerlerini konsola yazdırmaktadır. Değer atanmamış bileşenlerin (default) değerlerinin 0 olduğuna dikkat ediniz.

Array.cs

```
using System;

namespace Arrayler
{
    class Array05
    {
        static void Main()
        {
            int[] çarpan = new int[10];
            for (int i = 6; i < çarpan.Length; i++)
            {
                çarpan[i] = i * i;
            }
            for (int i = 0; i < çarpan.Length; i++)
            {
                Console.Write(çarpan[i]);
                Console.Write("\t");
            }
        }
    }
}
```

Aşağıdaki program da arrayin bileşenlerine bir döngü ile değer atamakta ve atanan değerleri başka bir döngü ile konsola yazdırmaktadır.

Array.cs

```
using System;

class Array01
{
    static void Main()
    {
        int[] intSayı = new int[5];
        for (int i = 0; i < intSayı.Length; i++)
            intSayı[i] = i * 10;
        for (int i = 0; i < intSayı.Length; i++)
            Console.WriteLine("intSayı[{0}] = {1}", i, intSayı[i]);
    }
}
```

Çıktı

```
intSayı[0] = 0
intSayı[1] = 10
intSayı[2] = 20
intSayı[3] = 30
intSayı[4] = 40
```

Seçkili (random) Erişim

Arrayin bileşenleri değişken olduklarından, onlara istendiğinde değer atanabileceğini, istenirse atanan değerlerin değiştirilebileceği açıktır. Yukarıdaki sicilNo[2] = 1785 atama deyimi, arraylerin üstün bir niteliğini ortaya koyar. Arrayin istenen bileşenine indeks sayısı ile doğrudan erişmek mümkündür. Her bileşen bir değişken olduğu için, o bileşen tipiyle yapılabilen her işlem onlar için de yapılabilir, değişkenlerle ilgili kurallar bileşenler için de aynen geçerlidir. SicilNo[2] = 1785 deyimi indeksi 2 olan bileşene 1785 değerini atamıştır. x aynı tipten bir değişken ise

```
x = sicilNo[2];
```

ataması, sicilNo[2] bileşeninin değerini x değişkenine aktarır; öyleyse, bu atama deyimi

```
x = 1785 ;
```

atamasına denktir. Aşağıdaki örnek, bileşenlerle işlem yapılabileceğini göstermektedir. Aylık ücretler ve gelir vergileri iki ayrı array ile tutulmakta, bir döngü ile %30 gelir vergisi hesaplanıp konsola yazılmaktadır.

```
using System;

namespace Arrayler
{
    class Array01
    {
        static void Main()
        {
            double[] aylıkÜcret = new double[3];
            aylıkÜcret[0] = 2782.45;
            aylıkÜcret[1] = 9346.74;
            aylıkÜcret[2] = 10867.83;

            double[] gelirVergisi = new double[3];
            for (int i = 0; i < aylıkÜcret.Length; i++)
            {
                gelirVergisi[i] = aylıkÜcret[i] * 30 / 100;

            Console.WriteLine("{0:c}  ücretin gelir vergisi =  {1:c} "
, aylıkÜcret[i], gelirVergisi[i]);
            }
        }
    }
}
```

Çıktı

```
2.782,45   TL  ücretin gelir vergisi =  834,74 TL
9.346,74   TL  ücretin gelir vergisi =  2.804,02 TL
10.867,83 TL  ücretin gelir vergisi =  3.260,35 TL
```

Bileşenlerden başka değişenlere veri aktarılabilir. Bileşenlerden veri aktarmalarında *istemli (explicit)* ve *istemsiz (implicit)* dönüşüm kuralları aynen geçerlidir. Aşağıdaki program `short` tipinden `int` tipine *istemsiz dönüşüm* yapılabileceğini göstermektedir.

Array.cs

```
using System;

namespace Arrayler
{
    class Array01
    {
        static void Main()
        {
            short[]  a = { 1, 2, 3, 4 };
            int[]  b = { 5, 6, 7, 8, 9 };
            Console.WriteLine(b[2]);
           b[2] = a[3];
            Console.WriteLine("b[2] = {0}" ,b[2]);
        }
    }
}
```

Çıktı

```
b[2] = 7
b[2] = 4
```

Aynı aktarmayı tersinden yapmak isteyelim. Yukarıdaki programda b[2] = a[3] deyimi yerine a[2] = b[3] yazarsak, derleyiciden şu hata mesajını alırız:

```
Error   1   Cannot implicitly convert type 'int' to 'short'. An explicit conversion exists (are you missing a cast?)...
```

Derleyicinin *istemsiz (implicit)* yapmadığı dönüşümü zorla yaptırmak istersek, *istemli dönüşüm* (implicit casting) yapabiliriz:

```
using System;

namespace Arrayler
{
    class Array01
    {
        static void Main()
        {
            short[] a = { 1, 2, 3, 4 };
            int[] b = { 5, 6, 7, 8, 9 };
            Console.WriteLine("a[2] = {0}", a[2]);
            a[2] = (short)b[3];
            Console.WriteLine("a[2] = {0}" , a[2]);
        }
    }
}
```

Çıktı

```
a[2] = 3
a[2] = 8
```

Arrayler çok sayıda değişken tanımlama ve bileşenlerine doğrudan istemli erişim sağlamaları yanında, döngüleri de çok kolaylaştırırlar. Aşağıdaki program `foreach` döngüsünün arraylere nasıl uygulanacağını göstermektedir.

Array.cs

```
using System;

namespace Arrayler
{
    class Array01
    {
        static void Main()
        {
            int[] arr = { 11, 12, 13, 14, 15, 16, -21, -11, 0 };
            foreach (int i in arr)
            {
                System.Console.Write("{0} \t " ,   i);
            }
        }
    }
}
```

Çıktı

```
11, 12, 13, 14, 15, 16, -21, -11, 0
```

Aşağıdaki program, bir metotla değer yazdırmakta, başka bir metotla değer okutmaktadır. Metotların parametreleri yaratılan array nesnedir.

Array.cs

```
using System;

namespace Arrayler
{
    class Array01
    {
        static int n = 10;
        static int[] arr = new int[n];

        static void değerYaz(int[] p)
        {
            for (int i = 0; i < p.Length; i++)
              p[i] = i;
        }

        static void değerOku(int[] r)
        {
            for (int i = 0; i < r.Length; i++)
                Console.WriteLine(r[i]);
        }
```

```
        static void Main()
        {
            değerYaz(arr);
            değerOku(arr);
        }
    }
}
```

Çıktı

```
0
1
2
3
4
5
6
7
8
9
```

Array Türleri

C# dilinde iki türlü array vardır. *Dikdörtgensel arrayler* ve *çentikli (jagged) arrayler.* Bunları örneklerle inceleyeceğiz.

Dikdörtgensel arrayler bir, iki ya da daha çok boyutlu olabilirler.

Bir Boyutlu Array

Bunlar tek indisli dizilerdir. 0-ıncı bileşenden başlayıp sonuncu bileşenine kadar index sırasıyla dizilirler. Yukarıda tanımladığımız arrayler ile aşağıdaki arrayler tek boyutludurlar.

```
int[]   derslik;
derslik = new int [3] {0, 1, 2};

string[]  kent = new string[5] {"Van", "Trabzon", "Kayseri", "Muş", "İzmir"};
string[]  kent = new string[] {"Van", "Trabzon", "Kayseri", "Muş", "İzmir"};
string[]  kent = {"Van", "Trabzon", "Kayseri", "Muş", "İzmir"};
```

Üçüncü satır, üç aşamalı array tanımını tek adımda yapmıştır. Dördüncü satır, dinamik uzunluklu array tanımıdır. Sonuncu satır, `new` kurucu deyimini kullanmadan array'i yaratmıştır.

`Length` arrayin uzunluğunu belirtir. Aşağıdaki program `Length` 'in işlevini göstermektedir.

Array01.cs

```
using System;

namespace Stringler
{
    class Array01
    {
        static void Main()
        {
            string[] kent = { "Van", "Trabzon", "Kayseri", "Muş",
```

```
"İzmir" };
            for (int i = 0; i < kent.Length; i++)
                Console.WriteLine("kent[{0}] = {1}", i, kent[i]);
        }
    }
}
```

Çıktı

```
kent[0] = Van
kent[1] = Trabzon
kent[2] = Kayseri
kent[3] = Muş
kent[4] = İzmir
```

Çok Boyutlu Array

Bileşenleri birden çok damgaya (index) bağlı olan arraylerdir. Aşağıdaki bildirimler, sırasıyla 1, 2 ve 3 boyutlu birer array bildirimidir.

```
int[,]   arr2Boyut ;       // 2 boyutlu array bildirimi
float[, ,]   arr3Boyut ;    // 3 boyutlu array bildirimi
float[, , ,]   arr4Boyut ;  // 4 boyutlu array bildirimi
```

Bir boyutlu arrayler için yaptığımız gibi, çok boyutlu arraylerin bileşenlerine de bildirim anında değer atayabiliriz. Aşağıdaki array, bileşenleri `int` tipi olan 2-boyutlu bir arraydir.

```
int[,] ikilSayı = new int[3, 2] { {1, 2}, {3, 4}, {5, 6} };
```

`ikilSayı` adlı arrayin bileşenlerini 3x2 tipi bir matris gibi yazdırabiliriz.

```
using System;

namespace Stringler
{
    class Array01
    {
        static void Main()
        {
            int[,] ikilSayı = new int[3, 2] { { 1, 2 }, { 3, 4 }, { 5,
6 } };
            Console.Write(ikilSayı[0, 0]); Console.Write("\t");
            Console.WriteLine(ikilSayı[0, 1]);
            Console.Write(ikilSayı[1, 0]); Console.Write("\t");
            Console.WriteLine(ikilSayı[1, 1]);
            Console.Write(ikilSayı[2, 0]); Console.Write("\t");
            Console.WriteLine(ikilSayı[2, 1]);


        }
    }
}
```

Çıktı

```
1       2
3       4
5       6
```

Arrayin bileşen sayısı matrisin bileşenlerinin sayısı kadardır. O halde, arrayin uzunluğu 3x2 = 6 dır; yani arrayin 6 bileşeni vardır. Bu matrise bakarak, arrayin bileşenlerini kolayca çıkarabiliriz.

Tek boyutlu arraylerin bileşenlerine olduğu gibi, çok boyutlu arraylerin bileşenlerine de seçkili (random) erişebiliriz. Örneğin, yukarıdaki programda

```
int[,] ikilSayı = new int[3, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 } };
```

deyimi yerine şunları koyabiliriz:

```
int[,] ikilSayı;
ikilSayı = new int[3, 2];
        ikilSayı[0, 0] = 1;
        ikilSayı[0, 1] = 2;
        ikilSayı[1, 0] = 3;
        ikilSayı[1, 1] = 4;
        ikilSayı[2, 0] = 5;
        ikilSayı[2, 1] = 6;
```

Görüldüğü gibi her bileşen iki damga (index) ile belirleniyor. Bu nedenle, bu tür arraylere 2-boyutlu array diyoruz. Aşağıdaki örnek `string` tipinden iki boyutlu bir arraydir.

```
string[,] adSoyad = new string[2, 2] { {"Melih","Cevdet"}, {"Orhan","Veli"} };
```

Bunun bileşenlerini 2x2 türünden bir matris olarak yazabiliriz.

Melih	Cevdet
Orhan	Veli

Arrayin bileşenleri

```
adSoyad[0,0] = Melih
adSoyad[0,1] = Cevdet
adSoyad[1,0] = Orhan
adSoyad[1,1] = Veli
```

olur. Bu bileşenleri iç-içe döngü ile yazdırabiliriz.

Array.cs

```
using System;

namespace Stringler
{
    class Array01
    {
        static void Main()
        {
            string[,] adSoyad = new string[3, 2] { { "Melih", "Cevdet"
```

```
}, { "Orhan", "Veli" } , {"Oktay", "Rıfat"}};
            int i=0, j=0;
            for (i = 0; i < 3; i++)
           {
                for (j = 0; j < 2; j++)
                    Console.Write("adSoyad[{0},{1}] = {2} \t ", i, j,
adSoyad[i, j]);
                Console.WriteLine();
           }
       }
   }
}
```

Aşağıdaki deyimler eşdeğerdir:

```
int[,] ary = new int [2,3] { {1,2,3}, {4,5,6} };
int[,] ary = new int [,]   { {1,2,3}, {4,5,6} };
int[,] ary =               { {1,2,3}, {4,5,6} };
```

Array'lerin bileşenlerinin matris biçiminde yazılışı birer dikdörtgen görüntüsü veriyor. O nedenle, bazı kaynaklar bir ve birden çok boyutlu array'lere dikdörtgensel array derler. Dikdörtgensel olmayan arrayleri biraz sonra açıklayacağız.

Tek boyutlularda olduğu gibi, çok boyutlu arraylerin bileşenlerine de program koşarken değer atayabiliriz. Aşağıdaki program, iki boyutlu bir arrayin bileşenlerine iç-içe iki döngü ile değer atamaktadır.

Array.cs

```
using System;

namespace Arrayler
{
    class Array01
    {
        static void Main()
        {
            int[,] arr = new int[5, 4];
            for (int j = 0; j < 4; j++)      // satır döngüsü
           {
                for (int i = 0; i < 5; i++) //kolon döngüsü
               {
                    arr[i, j] = i*j ;       // bileşenlere değer atar
                    Console.Write(arr[i, j]);
                    Console.Write("\t");
               }
                Console.WriteLine();
           }
       }
   }
}
```

Çıktı

```
0   0   0   0   0
0   1   2   3   4
```

```
0    2    4    6    8
0    3    6    9    12
```

Aşağıdaki program iki boyutlu bir array üzerinde foreach döngüsünün yapılışını göstermektedir.

Array.cs

```
using System;

namespace Arrayler
{
    class Array01
    {
        static void Main()
        {
            int[,] arr2B = new int[3, 2] { { 3, 9 }, { 4, 16 }, { 5, 25 } };
             foreach (int i in arr2B)
            {
                System.Console.Write("{0} ", i);
            }
        }
    }
}
```

Çıktı

```
3       9       4       16      5       25
```

Array Arrayi (çentikli array)

Her veri tipinden array yapılabilir demiştik. Array de bir veri tipidir. Öyleyse, bileşenleri arrayler olan array tanımlanabilir. Matematikteki fonksiyon fonksiyonu kavramına benzer. Bu tür arraylere çentikli array *(jagged array)* de denilir. Her bileşeni bir array olduğu ve o arraylerin uzunlukları farklı olabileceği için, arrayin bileşenlerini bir kağıda satır satır yazarsak, ortaya çıkacak görüntü bir dikdörtgen değil, girintili çıkıntılı bir şekildir. Çentikli array denmesinin nedeni budur.

Bunu daha iyi açıklayabilmek için, bileşenleri, sırayla,

```
{1, 3, 5}
{2, 4, 6, 8, 10}
{111, 222}
```

olan bir `arAr` arrayini düşünelim. Bunu, alıştığımız biçimde yazarsak

```
arAr[0] = {1, 3, 5}
arAr[1] = {2, 4, 6, 8, 10}
arAr[2] = {111, 222}
```

olur. Bir adım daha gidelim ve bileşenleri { } parantezi içine koyalım ve ortaya çıkan arrayi `arAr[][]` ile gösterelim.

```
ArAr[][]  =  { arAr[0] ,  arAr[1] ,  arAr[2]  }
          =  { {1, 3, 5} ,  {2, 4, 6, 8, 10} ,  {111, 222} }
```

arAr[][] parantezlerinin ilki (soldaki) en dıştaki {} parantezinin içindeki bileşen sayısını belirtir. İkinci (sağdaki) [] ise içteki parantezlerin (bileşenlerin) bileşen sayılarını belirtir.

Bu notasyonda anlaştıktan sonra, array arrayinin bildirimini kolayca yapabiliriz.

```
int[]  a = new int[] { 1, 3, 5 };
int[]  b = new int[] { 2, 4, 6, 8, 10 };
int[]  c = new int[] { 111, 222 };
int[][] arAr = new int[][] { a, b, c };
```

Bunların ilk üçü, sırasıyla, içteki arrayleri yaratır. Sonuncu deyim ise, yaratılan üç arrayi kendi bileşenleri yapan array arrayini yaratır. İstersek dört adımda yapılan bu işi tek adıma indirebiliriz.

```
int[][] arAr = new int[][] { new int[] { 1, 3, 5 }, new int[] { 2, 4,
6, 8, 10 }, new int[] { 111, 222 } };
```

Buna göre çentikli arrayimizi yaratan ve bileşenlerini konsola gönderen programı şöyle yazabiliriz.

arAr01.cs

```
using System;

namespace Arrayler
{
    class ArAr01
    {
        static void Main()
        {
            int[] a = new int[] { 1, 3, 5 };
            int[] b = new int[] { 2, 4, 6, 8, 10 };
            int[] c = new int[] { 111, 222 };
            int[][] arAr = new int[][] { a, b, c };

            // Bu dört adım yerine aşağıdaki deyim konulabilir
            //int[][] arAr = new int[][] { new int[] { 1, 3, 5 }, new
int[] { 2, 4, 6, 8, 10 }, new int[] { 111, 222 } };

            for (int r = 0; r < arAr[0].Length; r++)
            {
                for (int i = 0; i < arAr[r].Length; i++)
                {
                    Console.Write(arAr[r][i]);
                    Console.Write("\t");
                }
                Console.WriteLine();
            }
        }
    }
```

Çıktı

```
1       3     5
2       4     6     8     10
111     222
```

Aşağıdaki program array arrayini farklı bir yöntemle tanımlamaktadır.

Array.cs

```
using System;

using System;
```

```
namespace Arrayler
{
    class Array04
    {
        static void Main()
        {
            int[][] j = new int[3][];
            j[0] = new int[] { 1, 2, 3 };
            j[1] = new int[] { 1, 2, 3, 4, 5, 6 };
            j[2] = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

            for (int r = 0; r < 3; r++)
            {

                for (int i = 0; i < j[r].Length; i++)
                {
                    Console.Write(j[r][i]);
                    Console.Write("\t");
                }
                Console.WriteLine();
            }
        }
    }
}
```

Çıktı

```
1   2   3
1   2   3   4   5   6
1   2   3   4   5   6   7   8   9
```

Çıktıları görünce, array arraylerine neden çentikli dendiğini anlamış olmalısınız. Çentikli arraylerin bileşenleri bir tabloya dizildiğinde, dikdörtgensel bir biçim yerine, girintili çıkıntılı bir biçim almaktadırlar.

Alıştırmalar

Aşağıdaki deyimler, sırasıyla, tek boyutlu, çok boyutlu ve çentikli array'ler yaratır.

```
using System;


namespace Arrayler
{
    class TestArraysSınıfı
    {
        static void Main()
        {
            // Bir boyutlu array bildirimi
            int[] array1 = new int[5];

            // array'e bildirim anında değer atama
            int[] array2 = new int[] { 1, 3, 5, 7, 9 };

            // Alternatif sözdizimi
            int[] array3 = { 1, 2, 3, 4, 5, 6 };

            // 2 boyutlu array bildirimi
            int[,] multiDimensionalArray1 = new int[2, 3];
```

```
            // 2 boyutlu arraye bildirim anında değer atama
            int[,] multiDimensionalArray2 = { { 1, 2, 3 }, { 4, 5, 6 }
};

            // Çentikli array bildirimi
            int[][] çentikliArray = new int[6][];

            // ÇentikliArray'in ilk bileşenine atama
            çentikliArray[0] = new int[4] { 1, 2, 3, 4 };
        }
    }
}
```

Özet

Arrayler boyutlarına göre sınıflandırılırsa, bir array tek boyutlu ya da çok boyutlu olabilir.

Arrayler görünümlerine göre sınıflandırılırsa, bir array dikdörtgensel ya da çentikli olabilir.

Sayısal arraylerin öğelerinin öndeğerleri (default value) 0 dır. Referans öğelerin öndeğeri ise null'dır.

Çentikli array bileşenleri array olan arraydir. Dolayısıyla, bileşenleri referans tipindendir.

Arrayin bileşenleri 0 dan başlayarak numara sırası (indis, index) alır. İlk bileşeninin indisi 0 dır. n bileşenli bir arrayin sonuncu bileşeninin indisi (n-1) dir.

`[]` operatörü array'in bileşenlerini indisleriyle belirler.

Arrayin bileşenleri her veri tipinden olabilir. Array tipi de olabilir. Bunlara array arrayi ya da çentikli array denir.

Array tipleri referans tipindendir. Soyut Array tipinden yaratılmıştır. Ienumerable arayüzünü kullanır. Dolayısıyla, her array için foreach döngüsü kullanılabilir.

Array Sınıfı

Array yaratma, arraylerle işlem yapma, array içinde bileşen arama ve arrayi sıralama gibi arrayle ilgili işlemleri yapmaya yarayan öğeleri içeren bir sınıftır. Yedi tane özgeni (property) ve 30 dan fazla metodu vardır. Burada bir kaç örnek üzerinde duracağız:

Bir arrayin bileşen sayısını `Length` ya da `LongLength` özgenleri ile buluruz. Boyut sayısını `Rank` özgeni ile buluruz.

`Clear` metodu bir arrayin bileşen değerlerini siler. Sayısal bileşenleri `0`, referans tipleri `null` ve boolean tipleri `False` yapar.

`Clone` metodu bir arrayin kopyasını yapar.

`Copy` metodu bir arrayi aynı tipli başka bir array üzerine kopyalar.

`CreateInstance` metodu arrayin bir nesnesini yaratır.

`Equals` metodu arrayin iki nesnesinin eşit olup olmadığını söyler.

`Find` metodu array içinde aranan bir öğeyi bulur.

`GetValue` metodu arrayin istenen bir bileşeninin değerini bulur.

`IndexOf` metodu arrayin bir bileşeninin indisini bulur.

`Resize` metodu arrayin boyutunu değiştirir.

`Reverse` metodu tek boyutlu arrayin bileşenlerini ters sırada verir.

`SetValue` metodu bir bileşene değer atar.

`Sort` metodu bileşenleri sıralar

`ToString` metodu etkin nesneyi stringe dönüştürür.

Aşağıdaki program, Array sınıfının bazı metotlarının kullanılışını göstermektedir.

ArrayMetotları.cs

```
using System;
class ArrayMetotları
{
    static void Main()
    {
        // 5 bileşenli bir array yarat
        string[] adSoyad = new string[5];

        // 5 kişinin adını okut
        System.Console.WriteLine("Lütfen 5 kişinin adını giriniz:");
        for (int i = 0; i < adSoyad.Length; i++)
        {
            adSoyad[i] = System.Console.ReadLine();
        }

        // arrayin bileşenlerini giriş sırasıyla oku:
        System.Console.WriteLine("\nArray 'in orijinal sırası:");
        foreach (string ad in adSoyad)
        {
            System.Console.Write("{0}  ", ad);
        }

        // Arrayin sırasını tersine çevir:
        System.Array.Reverse(adSoyad);

        // Arrayi ters sırada yaz:
        System.Console.WriteLine("\n\nTers sıralı array:");
        foreach (string ad in adSoyad)
        {
            System.Console.Write("{0}  ", ad);
        }

        // Arrayi sırala (sort):
        System.Array.Sort(adSoyad);

        //  Sıralanmış arrayi yaz:
        System.Console.WriteLine("\n\n Sıralı Array:");
        foreach (string ad in adSoyad)
        {
            System.Console.Write("{0}  ", ad);
        }
    }
}
```

Çıktı

```
Lütfen 5 kişinin adını giriniz:
```

```
Hilâl
Esra
Ali
Güleda
Emrah

Array 'in orijinal sırası:
Hilâl  Esra  Ali  Güleda  Emrah

Ters sıralı array:
Emrah  Güleda  Ali  Esra  Hilâl

Sıralı Array:
Ali  Emrah  Esra  Güleda  Hilâl
```

Alıştırmalar

```
 using System;

class Uygulama
{
    static public void Main()
    {
        int[] number = { 1, 2, 3, 4, 5 };
        for (int i = 0; i < number.Length; i++) {
Console.WriteLine(number[i]); }
        for (int i = 0; i <= 4; i++) { Console.WriteLine(number[i]); }
        for (int i = 4; i >= 0; i--) { Console.WriteLine(number[i]); }
        foreach (int j in number) { Console.WriteLine(j); }
    }
}
```

ARRAY
Alıştırmalar

Tek Boyutlu array Bildirimi

Program 1:

Aşağıdaki program, array kullanmadan, 5 tane değişken tanımlayıp, onların değerlerini konsola yazıyor.

```
class Example {
public static void Main() {
int a=0, b=0, c=0, d=0, e=0, f=0;

a = 1;
b = 2;
c = 3;
d = 4;
e = 5;

if ( a == 1 )
System.Console.WriteLine("Value is 1\n");
if ( b == 2 )
System.Console.WriteLine("Value is 2\n");
if ( c == 3 )
System.Console.WriteLine("Value is 3\n");
if ( d == 4 )
System.Console.WriteLine("Value is 4\n");
if ( e == 5 )
System.Console.WriteLine("Value is 5\n");

}
}
```

.....

Aynı programı array kullanarak daha kolay ve kısa biçimde yazabiliriz.

Let us now re-write Listing 1 to use arrays.

Program 2

```
class Example {
public static void Main() {
int var=0;
```

```
int[] arr = new int[5] ;

for ( var = 0; var<5; var++ ) {
arr[var] = var + 1;
System.Console.WriteLine("Value is {0}\n",arr[var] );
}
}
}
```

```
using System;

class ArrayIntro
{
    public static void Main()
    {
        int[] iArray = new int[5];

        iArray[0] = 4;
        iArray[1] = 2;
        iArray[2] = 16;
        iArray[3] = 22;
        iArray[4] = 10;

        Console.WriteLine("index 4 = " + iArray[4]);
        Console.WriteLine("index 3 = " + iArray[3]);
        Console.WriteLine("index 2 = " + iArray[2]);
        Console.WriteLine("index 1 = " + iArray[1]);
        Console.WriteLine("index 0 = " + iArray[0]);
    }
}
```

Aynı array bildirimi aşağıdaki gibi de yapılabilir:

```
using System;

class ArrayIntro2
{
    public static void Main()
    {
        int[] iArray = {4, 2, 16, 22, 10};

        Console.WriteLine("index 4 = " + iArray[4]);
        Console.WriteLine("index 3 = " + iArray[3]);
        Console.WriteLine("index 2 = " + iArray[2]);
        Console.WriteLine("index 1 = " + iArray[1]);
        Console.WriteLine("index 0 = " + iArray[0]);
    }
}
```

Çentikli array Bildirimi:

Çentikli array arrayidir; yani her bileşeni bir array olan arraydir. Arraylerin uzunlukları birbirlerinden farklı olduğunda, alt alta dizildiğinde tablonun sağ tarafı girintili-çıkıntılı olur. Çentikli array (jagged array) denmesinin nedeni budur.

Aşağıdaki program, öğeleri (bileşenleri)
1, 2, 3, 4, 5
6, 7, 8
9, 10

Arraylerinden oluşan bir çentikli array bildirimi yapıyor ve onu consola yazdırıyor.

```
using System;

class JaggedIntro
{
    public static void Main()
    {
        int[][] iJagged;

        iJagged = new int[3][];

        iJagged[0] = new int[5];
        iJagged[1] = new int[3];
        iJagged[2] = new int[2];

        iJagged[0][0] = 1;
        iJagged[0][1] = 2;
        iJagged[0][2] = 3;
        iJagged[0][3] = 4;
        iJagged[0][4] = 5;
        iJagged[1][0] = 6;
        iJagged[1][1] = 7;
        iJagged[1][2] = 8;
        iJagged[2][0] = 9;
        iJagged[2][1] = 10;

        Console.WriteLine("array0 - {0}, {1}, {2}, {3}, {4}",
iJagged[0][0], iJagged[0][1],
            iJagged[0][2], iJagged[0][3], iJagged[0][4]);
        Console.WriteLine("array1 - {0}, {1}, {2}", iJagged[1][0],
iJagged[1][1], iJagged[1][2]);
        Console.WriteLine("array2 - {0}, {1}", iJagged[2][0],
iJagged[2][1]);
    }
}
```

```
using System;
public class MyArrayc2
{
 public static void Main()
 {
  int [][]arr=new int[4][];
  arr[0]=new int[3];
  arr[1]=new int[2];
  arr[2]=new int[5];
  arr[3]=new int[4];
```

```
  Console.WriteLine("Çentikli array için bileşen girişi:");

  for(int i=0 ; i < arr.Length ; i++)
  {
   for(int x=0 ; x < arr[i].Length ; x++)
   {
    String st= Console.ReadLine();
    int num=Int32.Parse(st);
    arr[i][x]=num;
   }
  }

  Console.WriteLine("");
  Console.WriteLine("Bileşenleri konsola yazdırma:");

  for(int x=0 ; x < arr.Length ; x++)
  {
   for(int y=0 ; y < arr[x].Length ; y++)
   {
    Console.Write(arr[x][y]);
    Console.Write("\0");
   }
   Console.WriteLine("");
  }
 }
}
```

Çok Boyutlu Array

Çok boyutlu arrayler, bileşenlerinin sayıları birbirlerine eşit olan arraylerin arrayidir. Başka bir deyişle, bileşen sayıları eşit olan arraylerin oluşturduğu özel bir çentikli (jagged) arraydir. Bileşen sayıları eşit olduğu için, arrayin bileşenleri alt alta yazıldığında ili boyt için dikdörtgen, üç boyut için dikdötrgenler prizması, ... vb şekiller oluşur. Bu nedenle, çok boyutlu arraylere, bazı kaynaklarda, dörtgensi arrayler (rectangular array) de denilir. Aşağıda iki boyutlu bir array bildirimi yapılmaktadır.

```
using System;

class MultiIntro
{
    public static void Main()
    {
        int[,] iMulti;

        iMulti = new int[2,4];

        iMulti[0,0] = 1;
        iMulti[0,1] = 2;
        iMulti[0,2] = 3;
        iMulti[0,3] = 4;

        iMulti[1,0] = 5;
        iMulti[1,1] = 6;
```

```
        iMulti[1,2] = 7;
        iMulti[1,3] = 8;

        Console.WriteLine("row0 - {0}, {1}, {2}, {3}", iMulti[0,0],
iMulti[0,1], iMulti[0,2],
            iMulti[0,3]);
        Console.WriteLine("row1 - {0}, {1}, {2}, {3}", iMulti[1,0],
iMulti[1,1], iMulti[1,2],
            iMulti[1,3]);
    }
}
```

Arrayler Üzerinde Döndüler

Arrayle r üzerinde döngü yapıları çok kolay ve kullanışlıdır. Aşağıdaki örnekler bunu kanıtlar.

For Döngüsü:

```
using System;

class ForIntro
{
    public static void Main()
    {
        for (int i = 0; i < 10; i++)
        {
            Console.WriteLine("Iteration " + i);
        }
    }
}
```

While Döngüsü:

```
using System;

class WhileIntro
{
    public static void Main()
    {
        int i = 0;

        while (i < 10)
        {
            Console.WriteLine("Iteration " + i);
            i++;
        }
    }
}
```

Do...While... Döngüsü:

```
using System;

class DoWhiIntro
{
    public static void Main()
    {
        int grade;

        do
        {
            Console.WriteLine("Input a grade between 0 and 100");
            grade = int.Parse(Console.ReadLine());
        } while (grade < 0 || grade > 100);
    }
}
```

---

foreach Döngüsü

```
using System;

class ForeaIntro
{
    public static void Main()
    {
        int[] iArray = new int[5];

        iArray[0] = 10;
        iArray[1] = 4;
        iArray[2] = 32;
        iArray[3] = 1;
        iArray[4] = 20;

        foreach (int i in iArray)
        {
            Console.WriteLine("i = " + i);
        }
    }
}
```

---

break ve continue Deyimleri

```
using System;

class BCIntro
{
    public static void Main()
    {
        int i = 0;

        while (i < 20)
        {
            i++;

            if (i != 0 && i % 5 == 0)
            {
                Console.WriteLine("");
```

```
                continue;
            }

            if (i / 4 == 3)
            {
                Console.WriteLine("Exiting...");
                break;
            }

            Console.Write("{0} ", i);
        }
    }
}
```

Bölüm 08

Program Akışının Yönlendirilmesi

blok,

if yönlendirmesi

switch yönlendirmesi

for döngüsü

while döngüsü

do ... while döngüsü

Bir program komutların yazıldığı sırada akar. Ama çoğunlukla, bu akışı yönlendirmek gerekir. Bu iş için denetim yapılarını kullanırız. Bunlar başlıca üç gruba ayrılabilir: Bloklar, Yönlendiriciler ve Döngüler. Bu yapılardan birisine girince, program o yapı içinde istenen her işi yapar. Bilgisayarın karmaşık işleri yapmasını sağlayan bu yapılar altı tanedir.

Bu yapılardan her birisi tek bir varlık olarak düşünülür, ama gerçekte her birisi birden çok deyim içeren birer yapıdır. Bunların her birisini örneklerle açıklayacağız.

Blok

Bir arada yürütülmesi istenen deyimleri içeren bir yapıdır. Bir blok içine giren program, aksi söylenmedikçe,o blok içindeki bütün deyimleri çalıştırır. Blok yapısının sözdizimi şöyledir:

```
{
      deyimler
}
```

{ } bloku içindeki deyimler istenildiği sayıda C# deyimlerinden oluşur. Hiç deyim içermeyen bloklara boş blok denir. İç-içe bloklar olabilir. Deyimler, basit C# deyimleri olabileceği gibi, yapısal deyimler de olabilir.

Örnekler

```
{
     System.Console.WriteLine("Ankara başkenttir. ");
     System.Console.WriteLine(x);
}
```

```
{  // Bu blok x ile y değişkenlerinin yerlerini değiştirir
     int yedek;       // blok içinde kullanılacak geçici değişken
     yedek = x;       // x değerini yedek adresine al
     x = y;           // y değerini x adresine al
     y = yedek;       // yedek değerini y nin adresine al
}
```

Blok içinde bildirimi yapılan bir değişken, o blok için bir yerel değişkendir. Bir blokun yerel değişkenine blok dışından erişilemez. Blok içinden dış bloka erişilebilir. Başka bir deyişle, blok içindeki bir operatör ya da metot, dış bloktaki değişkenleri ve metotları çağırabilir. Blokun işi bitince, yerel değişkene ayrılan adres yok olur. Bu nedenle, bir değişkenin erişilebirlik bölgesi (scope) o değişkenin tanımlandığı bloktur.

Yönlendiriciler

Program akışını, belli mantıksal koşullara göre istenen yöne saptıran denetim yapılarıdır. Bunlar da kendi içinde üçe ayrılır: if, if-else ve case yapıları.

if deyimleri

Program akışını, mantıksal koşullara (boolean) göre istenen yöne saptıran denetim yapılarıdır. Bu yapının üç türü vardır: tek seçenekli if, if-else ve çok seçenekli if.

1. Tek seçenekli if
2. if-else seçeneği
3. çoklu-durum seçeneği

Şimdi bunların her birisini örneklerle açıklayacağız.

Tek Seçenekli if

Bazı deyimlerin işlemesini, ancak belirli koşulların sağlanması durumunda isteyebiliriz. Bu durumda `if` yönlendirmesini kullanırız. Bu yönlendirmenin sözdizimi yapısı aşağıdaki iki durumdan birisi gibidir.

a. Eğer `if` denetiminden sonra bir tek deyim işleyecekse,

```
if ( mantıksal_deyim )
     deyim ;
```

biçimindedir.

b. Eğer `if` denetiminden sonra birden çok deyim işleyecekse, onlar bir blok ({ }) içine alınır.

```
if ( mantıksal_deyim )
       {
     deyimler;
       }
```

Eğer mantıksal_deyim true değerini alıyorsa deyim(ler) işlenir ve program if yapısından sonraki deyime geçer. Eğer mantıksal_deyim false değerini alıyorsa deyim(ler) işlenmeden atlanır ve program if yapısından sonraki deyimi işlemeye başlar.

IfYapısı01.cs

```
//if deyiminin kullanılışına örnektir
```

```
using System;

namespace DenetimYapıları
{
    class ifYapısı01
    {
        static void Main(string[] args)
        {
            string s;
            int n;

            Console.WriteLine("Bir tamsayı giriniz: ");
            s = Console.ReadLine();
            n = Int32.Parse(s);

            //if yönlendirmesi
            if (n > 0)
            Console.WriteLine("Girdiğiniz {0} sayısı pozitiftir. ", n);
        }
    }
}
```

Çıktı

```
Bir tamsayı giriniz:
45
Girdiğiniz 45 sayısı pozitiftir.
```

Bu program koşarken, ekrana önce "Bir tamsayı giriniz: " iletisi gelir. Bu iletiye karşılık, pozitif bir tamsayı girerseniz, örneğin 45, ekrana "`Girdiğiniz 45 sayısı pozitiftir.`" iletisi gelir. Eğer 0 ya da negatif bir tamsayı girerseniz, program girdiğiniz sayıyı string olarak okur, ve `int` tipine dönüştürür. Ama sayı pozitif olmadığı için if blokuna girmez. If blokundan sonraki deyime geçer. Ama `if` blokundan sonra başka deyim olmadığı için başka bir iş yapmadan program sona erer.

Eğer kullanıcı `int` tipinden başka bir değer girerse, `n = int.Parse(s)` tip dönüşümü olamayacağı için, program koşma hatası (runtime error) ile karşılaşır aşağıdaki uyarıyı vererek kesilir:

```
Unhandled Exception: System.FormatException: Input string was not in a correct
format...
```

Aşağıdaki program, if denetiminin bir mantıksal deyimi denetlediğini ve deyim true değerini alıyorsa if yapısına girildiğini göstermektedir.

```
/*
if denetimi için mutlaka bir mantıksal deyim (boolean) gereklidir.
*/
using System;

namespace DenetimYapıları
{
    class GeçersizIf
    {
        public static void Main()
        {
            if (1)
                Console.WriteLine("The if statement executed");
        }
    }
}
```

Bu program derleme anında şu hata mesajını verir:

Constant value '1' cannot be converted to a 'bool'

Bu ileti bize şunu söylüyor: Kaynak programdaki `if(1)` satırında sözdizimi hatası vardır. `if` anahtar sözcüğünden sonra mutlaka bir boolean deyim gelmelidir. Oysa, programda if sözcüğünden sonra yazılan (1) ifadesi bir boolean değildir. '1' booleana dönüştürülemez. *[C dilinde program yazanlar, bu biçimin o dilde geçerli olduğunu anımsayacaklardır. C# dili C dili üzerine kurulmuş olmakla birlikte, onun bazı niteliklerini değiştimiş, iyileştirmiştir. Dolayısıyla, C dilinde geçerli olan her şey C# dilinde geçerli değildir.]*

if-else yönlendirmesi

Bazı durumlarda önümüze iki seçenek çıkar. Belirli bir koşulun sağlanması durumunda seçeneklerden birinin, aksi halde ötekinin işlemesi istenebilir. Başka bir deyişle, bir koşulun sağlanıp sağlanmamasına bağlı olarak, iki seçenekten birisini mutlaka yaptırmak gerekir. Bu durumda, `if-else` yapısını kullanırız. Bu yapının sözdizimi şöyledir:

```
if ( mantıksal_deyim )
   deyim-1
      else
           deyim-2
```

Eğer `deyim-1` ve `deyim-2` yerinde işlenecek birden çok deyim varsa, onlar bir blok içine alınabilir:

```
 if ( mantıksal_deyim )
{
    deyimler-1
}
      else
      {
     deyimler-2
      }
```

Eğer mantıksal_deyim `true` değerini alıyorsa `deyim(ler)-1` işlenir ve program `if` denetim yapısından sonraki deyimi işlemeye başlar. Eğer `mantıksal_deyim false` değerini alıyorsa `deyim(ler)-1` işlenmeden atlanır ve `deyim(ler)-2` işlenir. Sonra, program `if` denetim yapısından sonraki deyimi işlemeye başlar. Bu yapıda program ya deyim(ler)-1 ya da deyim(ler)-2 'yi işler.

Aşağıdaki programları satır satır inceleyiniz. Her satırın işlevini algılayınız.

Program

```
using System;

namespace DenetimYapıları
{
    class IfElseYapısı
    {
        static void Main(string[] args)
        {
            string s;
            int i;

            Console.WriteLine("Bir tamsayı giriniz: ");
            s = Console.ReadLine();
            i = Int32.Parse(s);

            if (i > 0)
              Console.WriteLine("Girdiğiniz {0} sayısı pozitiftir ", i);
            else
```

```
            Console.WriteLine("Girdiğiniz {0} sayısı pozitif değildir ", i);
        }
    }
}
```

Aşağıdaki program verilen bir yılın artık yıl olup olmadığını bulur.

Program

```
//Metotların iç-içe kullanılışına örnektir.
using System;

namespace DenetimYapıları
{
    class ArtıkYıl
    {
        public static void Main()
        {
            int yıl;
            Console.WriteLine("Hangi yıl? (yyyy) :");
            yıl = Int32.Parse(Console.ReadLine());
            if ((yıl % 4 == 0 && yıl % 100 != 0) || yıl % 400 == 0)
                Console.WriteLine(" {0} yılı artık yıldır. Şubat ayı 29
çeker. ", yıl);
            else
                Console.WriteLine(" {0} yılı artık değildir. Şubat ayı 28
çeker. ", yıl);
        }
    }
}
```

Çıktılar

Programı

```
Hangi yıl? (yyyy) :
2007
 2007 yılı artık yıl değildir. Şubat ayı 28 çeker.

Hangi yıl? (yyyy) :
2008
 2008 yılı artık yıldır. Şubat ayı 29 çeker.
```

c. Çoklu-durum seçeneği

Bazan ikiden çok seçenek ortaya çıkabilir. Bu durumlarda `else-if` denen çoklu-durum yapısını kullanırız:

```
if (Koşul_1)
   Deyim_1;
else if (Koşul_2)
```

```
    Deyim_2;
else if (Koşul_3)
    Deyim_3;
...
else
    Deyim_n;
```

Else-ifYapısı01.cs

```
using System;

namespace DenetimYapıları
{
    class ElseIfYapısı01
    {
        static void Main(string[] args)
        {
            string s;
            int i;

            Console.WriteLine("Bugün ısı kaç derecedir? ");
            s = Console.ReadLine();
            i = Int32.Parse(s);

            if (i < 10)
                Console.WriteLine("Bugün hava soğuktur. ");
            else
                if (i < 20)
                    Console.WriteLine("Bugün hava serindir. ");
                else
                    Console.WriteLine("Bugün hava sıcaktır. ");
        }
    }
}
```

Else-ifYapısı02.cs

```
using System;

namespace DenetimYapıları
{
    class ElseIfYapısı02
    {
        static void Main(string[] args)
        {
            string s;
            int i;

            Console.WriteLine("Bir tamsayı giriniz ");
            s = Console.ReadLine();
            i = Int32.Parse(s);

            if (i %3 == 0)
                Console.WriteLine("{0} tamsayısı 3 ile tam bölünür. " , i
);
```

```
            else
                if (i %2 == 0)
                    Console.WriteLine("{0} tamsayısı çitf sayıdır. ",
i );
                else
                    Console.WriteLine("{0} tamsayısı tek sayıdır, ama 3
ile tam bölünmez. ", i );
        }
    }
}
```

Aşağıdaki programlar `else if` yapılarının istenildiği kadar iç-içe konulabileceğini göstermektedir.

Else-ifYapısı03.cs

```
using System;

namespace DenetimYapıları
{
    class ElseIfYapısı03
    {
        static void Main(string[] args)
        {
          string s;
          int i;

          Console.WriteLine("Not ortalamanızı giriniz : ");
          s = Console.ReadLine();
          i = Int32.Parse(s);

          if (i > 90)
            Console.WriteLine("Puanınız {0} ise notunuz A  olur. " , i );
            else
                if (i > 75)
            Console.WriteLine("Puanınız {0} ise notunuz B  olur. ", i);
                else
                    if (i > 60)
            Console.WriteLine("Puanınız {0} ise notunuz C  olur. ", i);
                    else
                        if (i > 50)
            Console.WriteLine("Puanınız {0} ise notunuz D  olur. ", i);
                        else
            Console.WriteLine("Puanınız {0} ise notunuz F  olur. ", i);
        }
    }
}
```

Else-ifYapısı04.cs

```
using System;

namespace DenetimYapıları
{
    public class ElseIfTest04
    {
        public static void Main()
        {
            Console.Write("Bir tuşa basınız: ");
            char c = (char)Console.Read();

            if (Char.IsUpper(c))
```

```
                Console.WriteLine("Büyük harf girdiniz.");
            else if (Char.IsLower(c))
                Console.WriteLine("Küçük harf girdiniz.");
            else if (Char.IsDigit(c))
                Console.WriteLine("Bir rakam girdiniz.");
            else
                Console.WriteLine("Alfasayısal olmayan bir karakter
girdiniz.");
        }
    }
}
```

Programın farklı koşmalarında, girilen karaktere bağlı olarak çıktılar şöyle olabilir.

```
Bir tuşa basınız: K
Büyük harf girdiniz.

Bir tuşa basınız: b
Küçük harf girdiniz.

Bir tuşa basınız: 7
Bir rakam girdiniz.

Bir tuşa basınız: %
Alfasayısal olmayan bir karakter girdiniz.
```

switch denetim yapısı

Program akışı bir yerde çok sayıda seçenekle karşılaşırsa, iç-içe `else if` denetimi yerine `switch` denetimi daha kolay olur. Sözdizimi şöyledir:

Sözdizimi (syntax)

```
switch (seçici)
{
   case seçki-1:
      deyim-1;
      sıçrama-1;

   case seçki-2:
      deyim-2;
      sıçrama-2
...
   [default:
      deyim
      sıçrama]
}
```

Bu yapıda case ifadelerinin sayısı için bir kısıt yoktur, gerektiği kadar case ifadesi konulabilir. Bu sözdiziminde geçen terimlerin açıklanması:

seçici

Seçici bir değişken veya bir ifade olabilir. Seçici değişken olduğunda `sbyte`, `byte`,`short`, `ushort`, `int`, `uint`, `long`, `ulong`, `char`, `string` ya da `enum` türlerinden herhangi birisinden bir `literal` (sabit değer) almalıdır. Seçici bir ifade ise bu türlerden literal bir değer veren formüldür. Burada formül, matematiksel formüllerde olduğu gibi, bir sonuç veren deyim olarak algılanmalıdır.

seçki

Seçici değişkenin veya seçici ifadenin aldığı bir değerdir. Bu değer tamsayı, char ya da string türünden bir literal (sabit değer) olmalıdır.

deyim

Seçici-değerine bağlı olarak yönlenilen bir seçenekte yapılacak iş(ler)i belirleyen deyim veya deyimlerden oluşan bir bloktur.

sıçrama

Bir seçeneğe yönlenen program, o seçenekteki işleri yapınca ya `switch` yapısının dışına çıkar ya da başka bir seçeneğe yönlendirilebilir. `Switch` yapısından çıkmak için

```
break;
```

deyimi kullanılır. `break` deyimi, akışı `switch` yapısından çıkarır ve switch yapısından sonraki ilk deyime gönderir.

Bazan bir seçenekteki işler bitince, akışı switch yapısından sonraki ilk deyime göndermek yerine başka bir seçeneğe göndermek gerekebilir. Bunun için

```
goto
```

deyimi kullanılır.

default

İsteğe bağlı bir seçenektir. Seçicinin değeri `case` ile belirlenen hiçbir seçki ile uyuşmadığı zaman, yapılmasını istediğimiz işler için gerekli deyim(ler)i bu seçeneğe yazarız. Hiçbir case yönlendirmesi gerçekleşemediğinde `default` seçenek çalışır. Seçicinin değeri mutlaka case'lerdeki seçkilerden birisine eşit olacağı garanti edilen durumlarda, `default` seçeneğini yazmanın gereği yoktur.

switch yapısının nasıl kullanıldığını aşağıdaki örnekte göreceğiz.

Switch01.cs

```
/*
switch-case yapısında seçici olarak string tipi değişken
kullanılmasına örnek.
*/
using System;

namespace DenetimYapıları
{
    class Switch01
    {
        [STAThread]
        static void Main(string[] args)
        {
            string ad, mesaj ;


            Console.Write("Adınız nedir? ");
            ad = Console.ReadLine();

            switch (ad)
            {
                case "Nihat":
                    mesaj = "En iyi golcümüz, değil mi?";
                    break;
```

```
                case "Arda":
                    mesaj = "İstikbalde umut veren bir oyuncu!";
                    break;
                case "Servet":
                    mesaj = "En iyi savunma oyuncumuz!";
                    break;
                case "Rüştü":
                    mesaj = "Her iyi kaleci gibi iyi hatalar yapıyor.";
                    break;
                default:
                    mesaj = "Volkan'a gelince, kafası yerine ellerini
kullandığında harika bir kaleci!";
                    break;
            }// switch sonu

            Console.WriteLine();
            Console.WriteLine(mesaj);
            Console.WriteLine();
        }// Main sonu
    }//class "switch01 sonu
}// namespace sonu
```

İpucu

Bunun için byte, short, int, long ya da char tiplerinden bir seçici değişken kullanılır. Float ve double tipinden seçici kullanılamaz. Seçici değişkenin alacağı sabit değerlere göre program akışı istenen seçeneğe sapar.

Burada herhangi bir `deyim-k` birden çok deyim içeriyorsa, onlar blok içine alınır ve tek deyimmiş gibi işlenir.

Switch02.cs

```
/*
switch-case yapısında seçici olarak int tipi değişken
kullanılmasına örnek.
*/
using System;

namespace DenetimYapıları
{
    class Switch02
    {
        public static void Main()
        {
            int ay;
            string s;
            Console.WriteLine("Kaçıncı ay ? ");
            s = Console.ReadLine();
            ay = Int32.Parse(s);

            switch (ay)
            {
                case 1: Console.WriteLine("Ocak"); break;
                case 2: Console.WriteLine("Şubat"); break;
                case 3: Console.WriteLine("Mart"); break;
                case 4: Console.WriteLine("Nisan"); break;
                case 5: Console.WriteLine("Mayıs"); break;
                case 6: Console.WriteLine("Haziran"); break;
                case 7: Console.WriteLine("Temmuz"); break;
```

```
                case 8: Console.WriteLine("Ağustos"); break;
                case 9: Console.WriteLine("Eylül"); break;
                case 10: Console.WriteLine("Ekim"); break;
                case 11: Console.WriteLine("Kasım"); break;
                case 12: Console.WriteLine("Aralık"); break;
            }
        }
    }
}
```

İpucu

Bu programda 17-inci satırdaki `ay = Int32.Parse(s);` atama deyimi yerine `ay = int.Parse(s);` deyimi konulabilir. Deneyerek görünüz. Bunun nedeni, C# dilindeki `int` tipinin `.NET Framework`'taki `Int32` tipi ile aynı olmasıdır. Visual Studio, C# dilinin veri tipleri yerine `.NET Framework`'taki karşılıklarının kullanılmasına izin verir.

Switch03.cs

```
/*
switch-case yapısında seçici olarak char tipi değişken
kullanılmasına örnek.
*/
using System;

namespace DenetimYapıları
{
    class Switch03
    {
        public static void Main()
        {
            char puan;
            string s;
            Console.WriteLine("Karne notunuz nedir?");
            s = Console.ReadLine();
            puan = char.Parse(s);


            switch (puan)
            {
                case 'A': Console.WriteLine("Pekiyi");
                    break;

                case 'B': Console.WriteLine("İyi");
                    break;

                case 'C': Console.WriteLine("Orta");
                    break;

                case 'D': Console.WriteLine("Hmmmm....");
                    break;

                case 'F': Console.WriteLine("Daha iyisini
başarabilirsin!");
                    break;

                default: Console.WriteLine("Başarı notun ne?");
                    break;
            }
        }
```

```
    }
}
```

Switch04.cs

```
using System;

namespace DenetimYapıları
{
    class Switch04
    {
        static void Main(string[] args)
        {
            string s;
            int i;

            Console.WriteLine("Not ortalamanızı giriniz : ");
            s = Console.ReadLine();
            i = Int32.Parse(s);

            switch (i / 10)
            {
                case 9:
                    Console.WriteLine("Notunuz A  dır. "); break;
                case 8:
                    Console.WriteLine("Notunuz B+  dir. "); break;
                case 7:
                    Console.WriteLine("Notunuz B-  dir. "); break;
                case 6:
                    Console.WriteLine("Notunuz C  dir. "); break;
                case 5:
                    Console.WriteLine("Notunuz D  dir. "); break;
                default:
                    Console.WriteLine("Notunuz F  dir. "); break;
            }
        }
    }
}
```

Aşağıdaki program bir otomatik çikolata makinasının çalışmasını sağlamaktadır. Makinaya 3 boy çikolata konulmaktadır: Küçük, Orta, Büyük. Bunların fiyatları, sırasıyla 250, 500 ve 750 YTL dir. Switch-case yapısını kullanan programda sıçrama goto deyimi ile yapılmaktadır. Programı inceleyiniz ve goto deyimlerinin oynadığı rolü algılayınız.

GoTo01.cs

```
/*
switch-case yapısında goto sıçramasının kullanılışına örnek.
*/
using System;

namespace DenetimYapıları
{
    class GotoYapısı
```

```
    {
        public static void Main()
        {
            Console.WriteLine("Hangi çikolata : 1=Küçük 2=Orta 3=Büyük");
            Console.Write("Lütfen seçiniz: ");
            string s = Console.ReadLine();
            int n = int.Parse(s);
            int bedel = 0;
            switch (n)
            {
                case 1:
                    bedel += 250;
                    break;
                case 2:
                    bedel += 250;
                    goto case 1;
                case 3:
                    bedel += 500;
                    goto case 1;
                default:
                    Console.WriteLine("Geçersiz seçim yaptınız. Lütfen 1,
2, 3 sayılarından birisini seçiniz.");
                    break;
            }
            if (bedel != 0)
                Console.WriteLine("Lütfen {0} YTL atınız.", bedel);
            Console.WriteLine("Afiyet olsun! Gene bekleriz!");
        }
    }
}
```

Çıktılar

Program 1, 2, 3 seçenekleri için ayrı ayrı koşturulursa, sırasıyla, aşağıdaki çıktılar görülecektir. Deneyiniz.

```
Hangi çikolata : 1=Küçük 2=Orta 3=Büyük
Lütfen seçiniz: 1
Lütfen 250 YTL atınız.
Afiyet olsun! Gene bekleriz!
```

```
Hangi çikolata : 1=Küçük 2=Orta 3=Büyük
Lütfen seçiniz: 2
Lütfen 500 YTL atınız.
Afiyet olsun! Gene bekleriz!
```

```
Hangi çikolata : 1=Küçük 2=Orta 3=Büyük
Lütfen seçiniz: 3
Lütfen 750 YTL atınız.
Afiyet olsun! Gene bekleriz!
```

Switch05.cs

```
//bir case seçeneğinden default seçeneğine sıçranamaz.
```

```
using System;

namespace DenetimYapıları
{
    class Switch05
    {
        public static void Main()
        {
            string day;
            Console.WriteLine("enter the day :");
            day = Console.ReadLine();
            switch (day)
            {
                case "Mon":
                    Console.WriteLine("day is Mon: go to work");
                    break;
                case "Monday":
                    Console.WriteLine("day is Monday: go to work");
                    break;
                default:
                    Console.WriteLine("default");
            }
        }
    }
}
```

Switch.06.cs

```
using System;
class SwitchTest1
{
    public static void Main()
    {
        int i = 1;
        switch (i)
        {
            case 1:
                Console.WriteLine("one");
                break;
            default:
                Console.WriteLine("default");
        }
    }
}
```

Bölüm 09

Döngüler

for döngüsü
do döngüsü
while döngüsü
foreach döngüsü

Belirli bir iş bir çok kez tekrarlanacaksa, programda bu iş bir kez yazılır ve döngü deyimleriyle istenildiği kadar tekrar tekrar çalıştırılabilir.

Bir döngüde, arka arkaya tekrarlanan deyimler döngü blokunu oluşturur. Bu deyimler birden çoksa { } bloku içine alınır. Bir döngü blokunda çok sayıda deyim olabileceği gibi, iç-içe döngüler de olabilir. Program akışının döngü blokunu bir kez icra etmesine döngünün bir adımı (bir tur) diyeceğiz.

C# dilinde döngü yapan dört ayrı yapı vardır. Aşağıdaki kesimlerde bu yapıları örneklerle açıklayacağız.

for Döngüsü

Bir deyim, önceden belirli olan sayıda tekrar edecekse, for döngüsünü kullanmak çok kolaydır. Önce basit bir örnekle başlayalım :

Aşağıdaki program aynı istenildiği kadar yazar.

ForLoop01.cs

```
using System;
namespace DenetimYapıları
{
    class ForLoop02
    {
        public static void Main()
        {
            for (int a = 5; a < 10; a++)
```

```
            {
                Console.WriteLine("C# ile programlama seçkin
programcılarin zevkidir.");
            }
        }
    }
}
```

Çıktı

```
C# ile programlama seckin programcilarin zevkidir.
C# ile programlama seckin programcilarin zevkidir.
C# ile programlama seckin programcilarin zevkidir.
C# ile programlama seckin programcilarin zevkidir.
C# ile programlama seckin programcilarin zevkidir.
```

Bu çıktıya dikkat edersek, for döngüsünün yapısının

```
            for (int a = 0; a < 10; a++)
            {

            }
```

blokundan oluştuğunu olduğunu görüyoruz. Döngü { } bloku içindeki deyimleri yürütür. Buna döngü bloku diyeceğiz. Burada `int a=0` döngü sayısını sayan sayaçtır, ilk değeri 0 dır. a <10 döngüye devam ya da dur diyecek mantıksal deyim (boolean) dir. a<10 olduğu sürece döngü tekrarlanacak, a>=5 olduğunda döngü duracaktır. a++ ise her döngünün her adımında sayacı 1 artıran deyimdir. Bu deyim, istenen sayıda adım atıldıktan sonra döngünün durmasını sağlar.

Şimdi bunlara göre, for döngüsünün söz dizimini yazabiliriz:

```
for (sayaç-ilk-değeri; döngü-koşulu; sayaç-değişimi)
{
deyimler
}
```

Bu yapıyı yukarıdaki örnekle karşılaştırarak söz dizimini kolayca algılayabilirsiniz. Zaten, aşağıdaki örnek programlarda da, konu yeterince tekrar edilecektir. Şimdilik şu özelikleri bilmek yeterlidir.

- Sayacın ilk değeri için döngünün ilk adımı (ilk tur) mutlaka çalışır.
- Sonraki adımların her birisi için önce döngü koşulu denetlenir, `true` değerini alıyorsa yeni tur atılır (döngü bloku işlenir).
- Döngü koşulu `false` değerini aldığında, program akışı döngünün dışına çıkar.

Elbette döngüleri kullanmaktaki amacımız, yukarıda yaptığımız gibi, aynı sözü defalarca söyletmek değildir. Onlarla harika işler yapabiliriz. O harika işlerin nasıl yapılabileceğini, aşağıdaki basit örneklerden öğreneceğiz.

ForLoop02.cs

```
// 5 den küçük pozitif tamsayıları yazar.
using System;
namespace DenetimYapıları
{
    class forLoop02
    {
```

```
        public static void Main()
        {
            for (int a = 0; a < 5; a++)
            {
                Console.WriteLine(a);
            }
        }
    }
}
```

Çıktı

```
0
1
2
3
4
```

Bu programda döngü blokunda bir tek Console.WriteLine(a) deyimi vardır. Böyle olduğunda { } parantezlerini kullanmayabiliriz.

```
            for (int a = 0; a < 5; a++)
            {
                Console.WriteLine(a);
            }
```

ForLoop03.cs

```
//10 'a kadar pozitif tamsayıları for döngüsü ile toplar
using System;
namespace DenetimYapıları
{
    class ForLoop03
    {
        public static void Main()
        {
            int sum = 0;
            for (int i = 0; i <= 10; i++)
            {
                sum = sum + i;

                Console.WriteLine("{0}ye kadar toplam: {1} ", i, sum);
            }
        }
    }
}
```

Çıktı

```
0ye kadar toplam: 0
1ye kadar toplam: 1
2ye kadar toplam: 3
3ye kadar toplam: 6
4ye kadar toplam: 10
5ye kadar toplam: 15
```

```
6ye kadar toplam: 21
7ye kadar toplam: 28
8ye kadar toplam: 36
9ye kadar toplam: 45
10ye kadar toplam: 55
```

break ve continue deyimleri

Bazı durumlarda, belli koşul oluşunca döngü adımları bitmeden döngüden çıkmak, bazı durumlarda da, döngünün bazı adımlarını işlememek gerekebilir. Bunu aşağıdaki iki deyimle yaparız.

break

Döngünün kesilmesine ve program akışının döngü blokunun dışına çıkmasına neden olur.

continue

Döngü bloku içinde kendisinden sonra gelen deyime geçmeden akışı durdurur, döngü başına gönderir ve döngünün bir sonraki adımının atılmasını sağlar.

`break` deyiminin etkisini aşağıdaki programda görebiliriz.

BreakKullanımı.cs

```
//döngü sayısını tamamlamadan döngüyü kesme yöntemini göstermektedir.
using System;
namespace DenetimYapıları
{
    class BreakKullanımı
    {
        public static void Main()
        {
            int faktoryel = 1;
            for (int n = 1; n <= 20; n++)
            {
                if (n == 6) break;
                faktoryel = faktoryel * n;
                Console.WriteLine("{0}! = {1} ", n, faktoryel);

            }
        }
    }
}
```

Çıktı

```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
```

Bu programı irdeleyelim. For döngüsünün koşullarına bakınca şunu anlıyoruz: Sayaç n=1 den başlayıp n<=20 olana kadar birer artarak gidecek ve böylece döngü 20 kez tekrarlanacak. Bu algımız doğrudur, ancak döngü blokunun içine baktığımızda

```
if (n==8) break;
```

deyimini görüyoruz. `break` deyimi döngünün kesilmesini ve program akışının döngü blokunun dışına çıkmasına neden olur. O halde, döngü, başlıkta istenenin aksine n==8 olduğunda kesilecektir. Dolayısıyla program koşturulunca ilk 7 sayının faktöryelleri yazılır.

Continue deyimi, break gibi döngüyü kesmez, yalnızca o an yapılan tekrarı tamamlatmaz ve sonraki tekrar için döngü başına döner.

`continue` deyiminin etkisini, yukarıdaki programın benzeri olan aşağıdaki programda görebiliriz.

ContinueKullanımı.cs

```
/*
continue deyimi, o anda blokta program akışını durdurur,
döngü başına gönderir, döngüyü sonraki adımdan yürütür.
*/
using System;
namespace DenetimYapıları
{
    class ContinueKullanımı
    {
        public static void Main()
        {
            int faktoryel = 1;
            for (int n = 1; n <= 10; n++)
            {
                if (n == 5) continue;
                faktoryel = faktoryel * n;
                Console.WriteLine("{0}! = {1} ", n, faktoryel);

            }
        }
    }
}
```

Çıktı

```
1! = 1
2! = 2
3! = 6
4! = 24
6! = 144
7! = 1008
8! = 8064
9! = 72576
10! = 725760
```

Bu çıktıda 5! olmadığını görüyoruz. Bunun nedeni şudur:

```
if (n == 5) continue;
```

deyimi, döngünün beşinci adımını tamamlatmadan program akışını durdurmuş ve döngü başına göndererek döngünün 6-ıncı adımdan devamını sağlamıştır.

Break ve continue deyimlerini amacımızı gerçekleştirmek için çok farklı biçimlerde kullanabiliriz. Aşağıdaki program her ikisini birden kullanmaktadır.

BreakContinue01.cs

```
//For döngüsünde break ve continue kullanımı
```

```
using System;
class BreakContinueTest
{
    public static void Main()
    {
        for (int i = 0; i < 20; ++i)
        {
            if (i == 10)
                break;
            if (i == 5)
                continue;
            Console.WriteLine(i);
        }
    }
}
```

Çıktı

```
0
1
2
3
4
6
7
8
9
```

Çıktıdan görüldüğü gibi, döngü 5 'e ulaştığında continue araya girmektedir. Dolayısıyla i=5 için Hiçbir iş yapılmadan, i=6 ya geçilip devam edilmektedir.

Öte yandan, döngü 20 adımda tamamlanacak olmakla birlikte, i=10 da break ile karşılaşmaktadır. Bu adımda döngü kesilmekte ve döngü blokunun dışına çıkılmaktadır.

ForLoop03.cs

```
// break ve continue kullanımı
using System;
namespace DenetimYapıları
{
    class ForLoop03
    {
        public static void Main()
        {
            for (int i = 0; i < 20; i++)
            {
                if (i == 10)
                    break;

                if (i % 2 == 0)
                    continue;

                Console.Write("{0} ", i);
            }
            Console.WriteLine();
        }
    }
}
```

Çıktı

```
1     3     5     7     9
```

Bu programı irdeleyelim. For döngüsünün koşullarına bakınca şunu anlıyoruz: Sayaç i=0 dan başlayıp i<20 olana kadar birer artarak gidecek ve döngü 20 kez tekrarlanacak iken

```
if (i==10) break;
```

deyimi, i==10 olduğunda döngüyü kesmekte ve program akışının döngü blokunun dışına çıkarmaktadır. Dolayısıyla, döngü, başlıkta istenenin aksine i==10 olduğunda kesilecektir.

Blok içindeki

```
if (i%2==0) continue ;
```

deyimine baktığımızda şunu anlıyoruz: `i%2==0` ise, yani i sayısı 2 ile tam bölünüyorsa, program akışı sonra gelen deyimleri yapmadan döngü başına dönüp döngüyü yeni adım için tekrarlamaktadır. Bu ise şu anlama gelir: i çift sayı olduğunda

```
Console.Write("{0} ", i);
```

deyimi çalışmadan döngü başına gidilecektir. Öyleyse, i çift sayı olduğunda ekrana bir çıktı yazılmayacak, yalnızca tek sayı olduğunda yazılacaktır. O nedenle, program yalnızca tek sayıları yazmıştır.

while döngüsü

Belirli bir mantıksal deyim sağlandığı sürece, belirli bir işin tekrarlanması isteniyorsa, bu denetim yapısı kullanılır. Bu yapıda, tekrarlamanın kaç kez olacağını önceden bilmek gerekmez. Sözdizimi şöyledir:

```
while (boolean)
{
deyimler
}
```

WhileLoop01.cs

```
//0 dan 9 kadar tamsayıları while döngüsü ile yazar
using System;

using System;
namespace DenetimYapıları
{
    class WhileLoop01
    {
        public static void Main()
        {
            int sayaç  = 0;

            while (sayaç < 10)
            {
                Console.Write("{0}    ", sayaç);
                sayaç++;
            }
            Console.WriteLine();
        }
    }
}
```

Çıktı

```
0    1    2    3    4    5    6    7    8    9
```

Bu programda (sayaç < 10) olduğu sürece, döngü bloku dediğimiz { } blok içindeki deyimler arka arkaya tekrar edilir. Bu tekrarların her birine döngünün bir adımı diyoruz. He adımda döngü sayacı 1 artar (sayaç++). Sonunda sayaç==10 olur ve döngü biter, program akışı, varsa döngü blokundan sonraki deyimden devam eder.

Bazı işleri birden çok döngü yapısıyla gerçekleştirebiliriz. Bu durumlarda, programcı birisini tercih etme hakkına sahiptir. Örneğin, önceki kesimde 1 den 10 'a kadar tamsayıların toplamını for döngüsü ile bulmuştuk. Aynı işi while, do-while ve foreach döngüleriyle de yapabiliriz.

While02.cs

```
using System;
namespace DenetimYapıları
{
    class while02
    {
        public static void Main()
        {
            int n = 1;
            int toplam = 0;

            while (n <= 10)
            {
                toplam = toplam + n;
                Console.WriteLine("TOPLAM[1...{0}] = {1} ", n, toplam);
                n++;
            }


        }
    }
}
```

Çıktı

```
TOPLAM[1...1] = 1
TOPLAM[1...2] = 3
TOPLAM[1...3] = 6
TOPLAM[1...4] = 10
TOPLAM[1...5] = 15
TOPLAM[1...6] = 21
TOPLAM[1...7] = 28
TOPLAM[1...8] = 36
TOPLAM[1...9] = 45
TOPLAM[1...10] = 55
```

do-while döngüsü

Sözdizimi şöyledir:

```
do
        {
        deyimler
        }
while (boolean) ;
```

Bu yapı while döngüsüne benzer; ama önce döngü blokundaki, { }, deyimler icra edilir, sonra (boolean) denetlenir. True değerini alırsa, program akışı döngünün başına döner ve bir tur daha atar. Tekrar (boolean) denetlenir true değerini alırsa yeni tura geçilir. Bu süreç (boolean) false değerini alana kadar devam eder.

Görüldüğü gibi do-while döngüsünde döngünün ilk adımı mutlaka icra edilir. While döngüsünden farkı da budur.

For ve while döngüleriyle yaptığımız toplama işlemini, aşağıda görüldüğü gibi, do-while döngüsüyle de yapabiliriz.

DoWhile01.cs

```
using System;
namespace DenetimYapıları
{
    class DoWhile01
    {
        public static void Main()
        {
            int n = 1;
            int toplam = 0;

            do
            {
                toplam = toplam + n;
                Console.WriteLine("TOPLAM[1...{0}] = {1} ", n, toplam);
                n++;
            } while (n <= 10);


        }
    }
}
```

Çıktı

```
TOPLAM[1...1] = 1
TOPLAM[1...2] = 3
TOPLAM[1...3] = 6
TOPLAM[1...4] = 10
TOPLAM[1...5] = 15
TOPLAM[1...6] = 21
TOPLAM[1...7] = 28
TOPLAM[1...8] = 36
TOPLAM[1...9] = 45
TOPLAM[1...10] = 55
```

DoWhile02.cs

```
using System;
namespace DenetimYapıları
{
    class DoWhile02
    {
        public static void Main()
        {
            string istek;

            do
            {
                // Bir menü yazar
                Console.WriteLine("Adres Defteri \n");

                Console.WriteLine("A - Yeni adres gir");
                Console.WriteLine("B - Adres sil");
                Console.WriteLine("C - Adres güncelle");
                Console.WriteLine("D - Adres gör");
                Console.WriteLine("Q - Çıkış \n");

                Console.WriteLine("Seçiminiz: (A,B, C, D, Q): ");

                // Kullanıcının isteğini oku
                istek = Console.ReadLine();

                // Kullanıcının isteğini yap
                switch (istek)
                {
                    case "A":
                    case "a":
                        Console.WriteLine("Yeni adres eklemek mi
istiyorsunuz?");
                        break;
                    case "B":
                    case "b":
                        Console.WriteLine("Bir adres silecek misiniz?");
                        break;
                    case "C":
                    case "c":
                        Console.WriteLine("Bir adres güncelleyecek
misiniz? ");
                        break;
                    case "D":
                    case "d":
                        Console.WriteLine("Bir adres mi göreceksiniz?");
                        break;
                    case "Q":
                    case "q":
                        Console.WriteLine("Hoşça kal!");
                        break;
                    default:
                        Console.WriteLine("{0} geçerli bir seçim
değildir", istek);
                        break;
                }

                // DOS ekranını bekleterek çıktının okunmasını sağlar
                Console.Write("Devam için bir tuşa basınız...");
                Console.ReadLine();
                Console.WriteLine();
```

```
                // Çıkış tusuna (Q) basılana kadar döngüyü tekrarlatan
boolean
            } while (istek != "Q" && istek != "q");
        }
    }
}
```

Çıktı

```
Adres Defteri


A - Yeni adres gir
B - Adres sil
C - Adres güncelle
D - Adres gör
Q - Çıkış

Seçiminiz: (A,B, C, D, Q):
B
Bir adres silecek misiniz?
```

foreach Döngüsü

Bir array'in ya da collection' ın öğeleri üzerinde döngü yapar.

Aşağıdaki program bir string array'in bileşenlerini yazmaktadır.

Program

```
//foreach loop
using System;
class ForEach
{
    public static void Main()
    {
        string[] a = { "Ankara", "İstanbul", "İzmir" , "Van" };
        foreach (string b in a)
            Console.WriteLine(b);
    }
}
```

Çıktı

```
Ankara
İstanbul
İzmir
Van
```

Bu programda şuna dikkat edilmelidir.

`foreach` döngüsünün kaç kez döneceğini belirten ifade `(string b in a)` dir. Bu ifade string tipinden `b` değişkenini tanımlamakta ve `b` değişkeni `a` array'inin öğelerine eşleşmektedir. Dolayısıyla, `a` array'inin bütün bileşenleri sırayla yazılmaktadır.

Bu program bir tamsayıda kaç hane olduğunu sayar.

HaneSay.cs

```
using System;

namespace DenetimYapıları
{
    class HaneSay
    {
        static void Main(string[] args)
        {
            string s;
            int  n = 0;
            Int32 i;

            Console.WriteLine("En çok 10 haneli bir tamsayı giriniz:");
            s = Console.ReadLine();
            i = Int32.Parse(s);

            do
            {
                ++n;
                i = i / 10;
            } while (i > 0);
            Console.WriteLine("{0} sayında {1} hane vardır.", i, n);
        }
    }
}
```

Çıktı

```
En çok 10 haneli bir tamsayı giriniz :
1234567890
1234567890 sayında 10 hane vardır.
Devam etmek için bir tuşa basın . . .
```

`do-while` döngüsü, döngü blokunu en az bir kez çalıştırır. Ondan sonra `while` koşulunu denetler.

Aşağıdaki program 6! değerini bulur.

```
using System;

namespace Döngüler
{
    class ForEach02
    {
        static void Main(string[] args)
        {
            int n = 1, w = 1;
            foreach (char c in "Ankara") { w = w * n; n++; }
            Console.Write(w);
        }
    }
}
```

Bölüm 10

Statik ve Dinamik Öğeler

Statik ve Dinamik Öğeler

Bir sınıfta static sıfatıyla nitelendirilen öğelerdir. Örneğin,

```
static int x;
static double ÜcretHesapla(){ ... };
```

deyimleri, sırasıyla, `int` tipinden `static x` değişkeni ile `double` değer alan `static ÜcretHesapla()` metodunun bildirimidir.

Bir sınıfın `static` öğeleri, o sınıfa ait bir nesne yaratılmadan erişilebilen öğelerdir. Sınıfın herhangi bir nesnesiyle bağlantısı olsun istenmeyen öğelere `static` nitelemesini vermek ona her yerden erişilebilen bir globallik niteliği verir. Daha önce, C# dilinde *global değişken* ve *global metot* olmadığını söylemiş, public nitelemesiyle öğelere bir tür global işlevsellik kazandırılabildiğini belirtmiştik. Öğelere globallik kazandırmanın ikinci etkin bir yolu `static` nitelemesidir. Statik öğeler, sınıfın bir nesnesi içinde olmadığından, onlara programın her yerinden erişilebilir. Örneğin, `Main()` metodu daima `static` nitelemesini alır. O nedenle, ait olduğu sınıfın bir nesnesi yaratılmadan doğrudan çalışabilir. Sınıfa ait bir ya da daha çok nesne yaratıldığında, bu nesneler içinde yapılan işler static öğeleri etkilemez. Bunun nedeni, statik bir öğeye, o sınıfa ait bir nesne yaratılmadan ana bellekte bir ve yalnız bir tane bellek adresi ayrılmasıdır. Nesnelerin yaratılıp yokedilmesi, ya da nesneler içindeki değişimler onları etkilemez. Dolayısıyla, örneğin, `static` bir değişkenin belli bir anda yalnız bir tane değeri olabilir. Daha önemlisi, ana bellekte nesneye bağlı olmadan birer adresleri olduğu için, static öğelere nesne yaratılmadan erişilebilir.

Statik olmayan öğelere *dinamik* öğeler diyeceğiz. Öndeğer (default) olarak, bildirim sırasında `static` nitelemesini almayan her öğe *dinamiktir*. Örneğin,

```
long y;
float Topla(a,b){ return a+b};
```

deyimleri, sırasıyla, `long` tipinden `dinamik x` değişkeni ile `float` değer alan dinamik `Topla()` metodunun bildirimidir.

Bir sınıfın dinamik öğelerine, o sınıfın yaratılan her bir nesnesi için, ana bellekte birer adres ayrılır. Başka bir

deyişle, bir sınıfın üç ayrı nesnesi yaratılmışsa, sınıftaki dinamik bir öğeye her nesnede bir tane olmak üzere üç ayrı bellek adresi ayrılır. Dolayısıyla, dinamik değişkenin yaratılan her nesnede ayrı bir değeri vardır. Ama, onların var olabilmeleri için, önce sınıfa ait nesnenin yaratılması gerekir.

Neden Nesne Yönelimli Programlama?

Bu soruya yanıt vermeden önce aşağıdaki bir dizi programı inceleyeceğiz. İncelemeyi bitirince, sorunun yanıtı kendiliğinden ortaya çıkacaktır. Örnek programların hepsi aynı işi yapmaktadırlar: Konsoldan brüt geliri okur, hesapladığı gelir vergisini konsola yazar.

İlk program, sözkonusu işi yapan en basit, en kısa ve en kolay programdır. Ama, yapısal programlama açısından asla tercih edilemeyecek kadar kötü bir programdır.

Program01.cs

```
using System;

namespace ErişimKısıtları
{
    class Program01
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Brüt Geliri giriniz :");
            Console.WriteLine(Decimal.Parse(Console.ReadLine())*40/100);
        }
    }
}
```

Çıktı

```
Brüt Geliri giriniz :
123456,789
49382,7156
```

Şimdi programımıza konsoldan veri okuyan, hesap yapan ve konsola veri yazan `VeriOku()`, `Hesapla()` ve `VeriYaz()` adlı üç ayrı metot ekleyelim. Ama öyle yapalım ki, `Main()` metodu `VeriYaz()` metodunu çağırsın. `VeriYaz()` metodu `Hesapla()` metodunu çağırsın. `Hesapla()` metodu `VeriOku()` metodunu çağırsın. Böylece arka arkaya dört metot çalışacaktır. Bir metodun başka bir metodu çağırması, matematikteki bileşik fonksiyon (fonksiyon fonksiyonu) kavramından başka bir şey değildir. İçteki fonksiyon, dıştaki fonksiyonun değişkenidir.

Program02.cs

```
using System;

namespace ErişimKısıtları
{
    class Program02
    {
        decimal VeriOku()
        {
            Console.WriteLine("Brüt Geliri giriniz :");
```

```
            return Decimal.Parse(Console.ReadLine());
        }

        decimal Hesapla()
        {
            return VeriOku() * 40 / 100;
        }

        void VeriYaz()
        {

            Console.WriteLine(Hesapla());
        }

        static void Main(string[] args)
        {
            VeriYaz();
        }
    }
}
```

Program derlenince aşağıdaki hata iletisini gönderir.

```
Error   1       An object reference is required for the non-static field, method, or property ...
```

Bu ileti bize, fonksiyonlara erişilemediğini, onların hangi nesnede olduklarını işaret eden referansların (işaretçi) verilmesi gerektiğini söylemektedir. Bu mesajı alınca şu iki yoldan birisini izleyebiliriz.

1. Metotlarımızı `static` yaparız.

2. Metotlarımızı içeren sınıftan bir nesne yaratırız (instantiate).

Her iki yöntemi ayrı ayrı deneyelim. Birincisi kolaydır. Metot bidirimlerinde her üçünü `static` sıfatı ile niteleyelim:

Program03.cs

```
using System;

namespace ErişimKısıtları
{
    class Program03
    {
        static decimal VeriOku()
        {
            Console.WriteLine("Brüt Geliri giriniz :");
            return Decimal.Parse(Console.ReadLine());
        }

        static decimal Hesapla()
        {
            return VeriOku() * 40 / 100;
        }

        static void VeriYaz()
```

```
        {

            Console.WriteLine(Hesapla());
        }

        static void Main(string[] args)
        {
            VeriYaz();
        }
    }
}
```

Çıktı

```
Brüt Geliri giriniz :
123456,789
49382,7156
```

Şimdi de ikinci yöntemi deneyelim. `Program03` sınıfından `p` adlı bir nesne yaratalım. Bunu yapmak için

```
Sınıf_adı nesne_adı = new Sınıf_adı();
```

nesne yaratıcı (instantiate) deyimini kullanıyoruz. Örneğimizde, nesne_adı p olacaktır.

Program04.cs

```
using System;

namespace ErişimKısıtları
{
    class Program04
    {
        decimal VeriOku()
        {
            Console.WriteLine("Brüt Geliri giriniz :");
            return Decimal.Parse(Console.ReadLine());
        }

        decimal Hesapla()
        {
            return VeriOku() * 40 / 100;
        }

        void VeriYaz()
        {

            Console.WriteLine(Hesapla());
        }

        static void Main(string[] args)
        {
            Program04 p = new Program04();
            p.VeriYaz();
        }
    }
}
```

Çıktı

```
Brüt Geliri giriniz :
123456,789
49382,7156
```

İpucu

Buradaki `VeriOku()`, `Hesapla()`, `VeriYaz()` metotlarının önüne `static` nitelemesi konulmadığı için, doğal olarak (default) *dinamik* olurlar; yani static değildirler. Dolayısıyla, ancak sınıfa ait bir nesne yaratılınca o nesne içinde ana bellekte kendilerine birer yer ayrılır. Üstlendikleri işlevlerini o nesne içinde yaparlar. Böyle olduğunu görmek için, `p` ve `q` adlarıyla iki farklı nesne yaratalım ve o nesneleredeki metotlarımızı farklı verilerle çalıştıralım. `p` ve `q` nesnelerinden gelen çıktıların farklı olduğunu görebiliriz.

Program05.cs

```
using System;

namespace ErişimKısıtları
{
    class Program05
    {
        decimal VeriOku()
        {
            Console.WriteLine("Brüt Geliri giriniz :");
            return Decimal.Parse(Console.ReadLine());
        }

        decimal Hesapla()
        {
            return VeriOku() * 40 / 100;
        }

        void VeriYaz()
        {

            Console.WriteLine(Hesapla());
        }

        static void Main(string[] args)
        {
            Program05 p = new Program05();
            Program05 q = new Program05();

            p.VeriYaz();
            q.VeriYaz();
        }
    }
}
```

Çıktı

```
Brüt Geliri giriniz :
123456,789
49382,7156
Brüt Geliri giriniz :
2000
800
```

Şimdi başka bir durumu deneyelim. Sınıfta bildirimi yapılan metotların hepsini satic yapalım ve sınıftan bir nesne yaratmayı deneyelim.

Program06.cs

```
using System;

namespace ErişimKısıtları
{
    class Program06
    {
        static decimal VeriOku()
        {
            Console.WriteLine("Brüt Geliri giriniz :");
            return Decimal.Parse(Console.ReadLine());
        }

        static decimal Hesapla()
        {
            return VeriOku() * 40 / 100;
        }

        static void VeriYaz()
        {

            Console.WriteLine(Hesapla());
        }

        static void Main(string[] args)
        {
            Program06 p = new Program06();
            p.VeriYaz();

        }
    }
}
```

Bu programı derlemek istersek şu hata mesajını verecektir:

```
Error 1 Member 'ErişimKısıtları.Program06.VeriYaz()' cannot be accessed with an instance reference; qualify it with a type name instead...
```

Mesaj bize şunu söylüyor. p referanslı (p nin işaret ettiği) bir nesne yaratılmıştır ve p işaretçisi yaratılan bu nesnenin adresini işaret etmektedir. Sınıfın static öğelerinin bellek adreslerini göstermez. Dolayısıyla, p referansı ile static `VeriYaz()` metoduna erişilemez. Çünkü statik öğelere ana bellekte ayrılan yer p nin işaret ettiği nesne içinde değildir. p işaretçisi ancak yaratılan nesne içindeki dinamik öğeleri gösterebilir.

İpucu

Şimdi sırayla aşağıdaki durumları ayrı ayrı deneyelim:

Metotların üçü de dinamik ise, program derlenir ve çalışır.

Metotların üçü de static ise, en sondaki `p.VeriYaz()` deyimi yerine `VeriYaz()` ya da `Program06.VeriYaz()` yazılırsa program derlenir ve çalışır. Bu durumda yaratılan nesne işleme girmez.

`VeriYaz()` ve `Hesapla()` metotları dinamik, `VeriOku()` metodu statik ise program derlenir ve çalışır.

`VeriYaz()` dinamik, `Hesapla()` static, `VeriOku()` dinamik ise program derlenemez.

`VeriYaz()` metodu static yapılınca, öteki ikisi ne olursa olsun, program derlenemez.

Şaşırtıcı görünen bu durumun basit bir nedeni vardır. O nedeni artık biliyoruz. Dinamik öğeye referans ile erişilir. VeriYaz() metodu dinamik yapılınca, p referansı onun yerini işaret eder. VeriYaz() metoduna erişilince, artık kontrol p nin elinden çıkar, VeriYaz() metoduna geçer. VeriYaz() metodu Hesapla() metodunu çağırıyor. Eğer, `Hesapla()` dinamik ise, `VeriYaz()` metodunun bulunduğu nesne içindedir ve oraya erişebilir. Eğer `Hesapla()` statik ise, zaten ona erişmek mümkündür. Hesapla `VeriOku()` metodunu çağırıyor. Bu metot statik ise, yeri bellidir ve ona erişir. Ama VeriOku() dinamik ise, onun bulunduğu nesneyi Hesapla() metodu göremez.

Çıktı

```
Brüt Geliri giriniz :
123456,789
49382,7156
Brüt Geliri giriniz :
2000
800
```

Erişimkısıtları.cs

```
using System;

namespace ErişimKısıtları
{
    class Program01
    {
        static void Main(string[] args)
        {
            Vergiler.VergiYaz();
        }
    }
}

class Vergiler
```

```
{
    static decimal brütGelir;
    const byte vergiYüzdesi = 40;

    public static decimal GelirVergisiHesapla(decimal x)
    {
        decimal y;
        brütGelir = x;
        y = x * vergiYüzdesi/100;
        return y;
    }

    public static void VergiYaz()
    {
        string s;
        Console.WriteLine("Brüt Geliri giriniz :");
        s = Console.ReadLine();
        Vergiler.brütGelir = Decimal.Parse(s);
        Console.WriteLine(GelirVergisiHesapla(Vergiler.brütGelir));
    }
}
```

Çıktı

```
Brüt Geliri giriniz :
3425,467
3425,467 nin vergisi 1370,1868 YTL dir.
```

Program02.cs

```
using System;

namespace ErişimKısıtları
{
    class Program01
    {
        static void Main(string[] args)
        {
            Vergiler.VergiYaz();
        }
    }
}


class Vergiler
{
    static decimal brütGelir;
    const byte vergiYüzdesi = 40;

    public static decimal GelirVergisiHesapla(decimal x)
    {
        decimal y;
        brütGelir = x;
        y = x * vergiYüzdesi/100;
```

```
        return y;
    }

    public static void VergiYaz()
    {
        string s;
        decimal vergi;
        Console.WriteLine("Brüt Geliri giriniz :");
        s = Console.ReadLine();
        Vergiler.brütGelir = Decimal.Parse(s);
        vergi = GelirVergisiHesapla(Vergiler.brütGelir);
        Console.WriteLine("{0} nin vergisi {1} YTL dir." , brütGelir,
vergi);
    }

}
```

Çıktı

```
Brüt Geliri giriniz :
3000
3000 nin vergisi 1200 YTL dir.
```

Program03.cs

```
using System;

namespace ErişimKısıtları
{
    class Program01
    {
        static void Main(string[] args)
        {
            Hesap.VergiYaz();
        }
    }
}


class Vergiler
{
    public static decimal brütGelir;
    const byte vergiYüzdesi = 40;

    public static decimal GelirVergisiHesapla(decimal x)
    {
        decimal y;
        brütGelir = x;
        y = x * vergiYüzdesi/100;
        return y;
    }
}

class Hesap
```

```
{
    static decimal vergi;
    public static void VergiYaz()
    {
        string s;

        Console.WriteLine("Brüt Geliri giriniz :");
        s = Console.ReadLine();
        Vergiler.brütGelir = Decimal.Parse(s);
        vergi = Vergiler.GelirVergisiHesapla(Vergiler.brütGelir);
        Console.WriteLine("{0} nin vergisi {1} YTL dir." ,
Vergiler.brütGelir, vergi);
    }
}
```

Çıktı

```
Brüt Geliri giriniz :
3000
3000 nin vergisi 1200 YTL dir.
```

Bölüm 11

Erişim Belirteçleri

(Access Modifiers)

Erişim Belirteci Nedir?

C# dili nesne yönelimli bir dil olduğundan, her şey sınıflar içinde tanımlanır. Sınıflara ve sınıf öğelerine erişim kısıtlanabilir ya da belli düzeylerde erişime izin verilebilir. Öğelere erişimi kıstlayan ya da yetki veren anahtar sözcüklere *Erişim Belirteçleri* (access modifiers) denir.

Erişim belirteçleri, bir sınıfa ya da bir sınıfa ait öğelere erişilebilme kısıtlarını veya yetilerini belirleyen anahtar sözcüklerdir. Esas olarak dört tane erişim belirteci vardır. Aşağıdaki tablonun beşinci satırında yazılan belirteç bir bileşiktir.

Erişim Belirteçleri

public	Erişim kısıtı yoktur; her yerden erişilir
protected	Ait olduğu sınıftan ve o sınıftan türetilen sınıflarından erişilebilir

internal	Etkin projeye ait sınıflardan erişilebilir, onların dışından erişilemez
private	Yalnız bulunduğu sınıftan erişilir, dıştaki sınıflardan erişilemez
protected internal	Etkin projeye ait sınıflardan ve onların türevlerinden erişilebilir

`Bir öğe`, protect internal hariç, öteki erişim belirteçlerinden yalnız birisini alabilir.

`namespace` erişim belirteci almaz; çünkü o daima `public` nitelemelidir.

`Sınıflar` public ya da internal nitelemesini alabilirler; ama protected ile private nitelemesi alamazlar.

`enum` erişim belirteci almaz; çünkü o daima `public` nitelemelidir.

`Bir öğe`, bildirildiği ortama bağlı olarak, yalnız izin verilen erişim belirteçlerini alabilir. Eğer erişim belirteci almamışsa, öndeğer (default) belirteç etkin olur.

Başkalarının içinde yuvalanmamış üst-düzey tipler ancak `internal` ve `public` nitelemesini alabilirler. Bu tipler için öndeğer (default) niteleme `public`'tir.

Oğul sınıflar, doğal olarak *ata*'nın niteleme kısıtlarına tabidir. İstenirse daha çok kısıt konabilir; ama kısıtlar azaltılamaz.

`İç-içe` yuvalanmış tipler aşağıdaki tabloda gösterilen erişim nitelemelerini alabilirler.

Aidiyet	Öndeğer (default erişim belirteci	Nitelenebilen erişim belirteci
enum	public	Hiç
class	private	public protected internal private protected internal
interface	public	Hiç
struct	private	public internal private

Aşağıdaki örnek public ve private öğelere erişimi göstermektedir.

Access01.cs

```
// Public ve private erişim
```

```
using System;

class Ev
{
    private int oda;    // private nitelemeli
    int dolap;          // doğal private erişimli
    public int kapı;    // public erişimli

    // oda ve dolap Ev sınıfından erişilebilir
    public void setOda(int a)
    {
        oda = a;
    }

    public int getOda()
    {
        return oda;
    }

    public void setDolap(int a)
    {
        dolap = a;
    }

    public int getDolap()
    {
        return dolap;
    }
}

public class Uygulama
{
    public static void Main()
    {
        Ev birEv = new Ev();

    /* oda ve dolap değişkenlerine

         ancak metotlarla erişilir */

      birEv.setOda(5);

      birEv.setDolap(7);

      Console.WriteLine("birEv.oda   : " + birEv.getOda());

      Console.WriteLine("birEv.dolap : " + birEv.getDolap());

      //  oda ve dolap değişkenlerine aşağıdaki gibi erişilemez:

      //  birEv.oda = 10;    // Hata! oda  private'dir!

      //  birEv.dolap = 9;   // Hata! dolap  private'dir!
```

```
        // kapı public olduğu için doğrudan erişilir.

        birEv.kapı = 8;

        Console.WriteLine("birEv.kapı  : " + birEv.kapı);
    }
}
```

Aşağıdaki örnek `protected` nitelemeli öğelere oğul'dan erişilebildiğini göstermektedir.

Access02.cs

```
/*
 * protected nitelemeli öğelere sınıf içinden ve
 * oğul'dan erişilebilir.
*/
using System;

class Ata
{
    protected int m, n;

  // m ile n Ata ye göre private gibidir,

  // ama onlara Oğul 'den erişilebilir

    public void Set(int a, int b)
    {
        m = a;
        n = b;
    }

    public void Göster()
    {
        Console.WriteLine(m + " " + n);
    }
}

class Oğul : Ata
{
    int çarpım; // private

    // Oğul den Ata'deki m ve n ye erişilir
    public void Set()
    {
        çarpım = m * n;
    }

    public void Yaz()
    {
        Console.WriteLine(çarpım);
    }
}

public class ProtectedDemosu
{
```

```
    public static void Main()
    {
        Oğul birD = new Oğul();

        birD.Set(2, 3);  // Oğul 'den erişiliyor
        birD.Göster();   // Oğul 'den erişiliyor

        birD.Set();      // Oğul 'nin öğesi
        birD.Yaz();      // Oğul 'nin öğesi
    }
}
```

Aşağıdaki programdaki `Hesap` sınıfı birisi `public`, ötekisi `private` nitelemeli iki metot içeriyor. Uygulama sınıfındaki `Main()` metodu her ikisini çağırıyor. Ama derleyici `private` olanın çağrılmasına izin vermiyor. Program derleme hatası veriyor.

Access03.cs

```
using System;
class Uygulama
{
    static void Main()
    {
        Hesap obj = new Hesap();
        System.Console.WriteLine("2 + 3 = {0}", obj.Topla(2, 3));
        //Bu deyim derlenemez!!
        System.Console.WriteLine("3 - 2 = {0}", obj.Çıkar(3, 2));
    }
}

class Hesap
{
    public long Topla(int a, int b)
    {
        return a + b;
    }

    private long Çıkar(int c, int d)
    {
        return c - d;
    }
}
```

```
Error   1   'Hesap.Çıkar(int, int)' is inaccessible due to its protection level..
```

Şimdi bu basit programı düzeltip (debug) çalışır duruma getirmek için neler yapabileceğimizi düşünelim. Bunları düşünüp denedikçe, erişim belirteçlerinin işlevlerini daha iyi kavrayacağız.

1. Derleyicinin itiraz ettiği erişim kısıtını kaldırabiliriz. `Çıkar()` metodunun nitelemesini `private` yerine `public` yaparsak, programın derlendiğini ve çalıştığını görebiliriz.

```
    public long Çıkar(int c, int d)
```

2. `internal` nitelemesi etkin olan programa ait sınıflardan (assembly) erişilmesine izin verdiğine göre, `Çıkar()` metodunun nitelemesini `private` yerine `internal` yaparsak, programın derlendiğini ve çalıştığını görebiliriz.

```
internal long Çıkar(int c, int d)
```

3. `protected internal` nitelemesi etkin projeye ait sınıflardan ve onların türevlerinden erişilmesine izin verdiğine göre, `Çıkar()` metodunun nitelemesini `private` yerine `protected internal` yaparsak, programın derlendiğini ve çalıştığını görebiliriz.

```
protected internal long Çıkar(int c, int d)
```

4. `protected internal` nitelemesi etkin projeye ait sınıflardan ve onların türevlerinden erişilmesine izin verdiğine göre, `Çıkar()` metodunun nitelemesini `private` yerine `protected internal` yaparsak, programın derlendiğini ve çalıştığını görebiliriz.

```
protected internal long Çıkar(int c, int d)
```

5. `protected internal` nitelemesi yalnızca ait olduğu sınıftan ve o sınıftan türetilen sınıflarından erişilmesine izin verdiğine göre, `Çıkar()` metodunun nitelemesini `private` yerine `protected` yaparsak, programın derlenemediğini görürüz.

```
protected long Çıkar(int c, int d)
```

6. `private` nitelemeli öğelere aynı sınıf içinden erişilebildiğini biliyoruz. O halde, `Main()` metodunu `Çıkar()` metoduyla aynı sınıfa koyarsak, programımız çalışır duruma gelecektir:

```
using System;

class Hesap
{
    public long Topla(int a, int b)
    {
        return a + b;
    }

    private long Çıkar(int c, int d)
    {
        return c - d;
    }

    static void Main()
    {
        Hesap obj = new Hesap();
        System.Console.WriteLine("2 + 3 = {0}", obj.Topla(2, 3));
        //Bu deyim derlenenir!!
        System.Console.WriteLine("3 - 2 = {0}", obj.Çıkar(3, 2));
    }
```

```
}
```

7. `Main()` metodunu `Çıkar()` metodunu içeren `Hesap` sınıfının bir alt sınıfına koyarsak, programımız çalışmaya devam edecektir:

```
using System;

class Hesap
{
    public long Topla(int a, int b)
    {
        return a + b;
    }

    private long Çıkar(int c, int d)
    {
        return c - d;
    }

    class Uygulama
    {
        static void Main()
        {
            Hesap obj = new Hesap();
            System.Console.WriteLine("2 + 3 = {0}", obj.Topla(2, 3));
            //Bu deyim derlenemez!!
            System.Console.WriteLine("3 - 2 = {0}", obj.Çıkar(3, 2));
        }
    }
}
```

Alıştırma

Aşağıdaki programda `Uygulama` sınıfındaki `Main()` metodu `AAA` sınıfındaki `bbb()` metodunu çağırmak istiyor. Ancak derleyici `bbb()` metoduna erişilemediği için hata iletisini veriyor.

```
using System;

class Uygulama
{
    public static void Main()
    {
        AAA.bbb();
    }
}
class AAA
{
    static void bbb()
    {
        System.Console.WriteLine("AAA.bbb()");
    }
```

```
    public static void ccc()
    {
        System.Console.WriteLine("AAA.ccc()");
        bbb();
    }
}
```

```
Error   1       'AAA.bbb()' is inaccessible due to its protection level ...
```

Programı debug edip çalışır duruma getirmek için aşağıdakilerden hangisi programı çalıştırır? Size göre, programı çalıştıran seçeneklerden hangisi ne zaman en uygundur?

1. `bbb()` metoduna public nitelemesini ekleriz:

```
static public void bbb()
```

2. `bbb()` metoduna internel nitelemesini ekleriz:

```
static internal void bbb()
```

3. `AAA` sınıfına ait bir nesne yaratıp, `bbb()` metodunu nesne içinden çağırırız. (Bunun için gerekli kodları yazınız.)

4. `Main()` metodunu `AAA` sınıfı içine alırız.

5. `Uygulama` sınıfını `AAA` sınıfı içine alırız.

6. `AAA` sınıfını `Uygulama` sınıfı içine alırız.

7. `bbb()` metodunun bildirimini `Uygulama` sınıfı içine alırız.

8. `bbb()` metodunun bildirimini `Main()` metodu içine alırız.

Bölüm 12

Metotlar

Metot Nedir?

Her bilgisayar dili değişken ve fonksiyon kavramlarına sahiptir. Bu tesadüfen değil, işin doğasından kaynaklanan bir durumdur. Değişkenler, bilgisayar programının işleyeceği ham maddedir. Bu ham maddelerin işlenip ürün haline getirilmesi için kullanılan araçlar fonksiyonlardır.

Nesne Yönelimli (Object Oriented- OO) Programlama Dillerinde her şey sınıflar (`class`) içinde tanımlanır. Bir sınıfa ait değişkenler o sınıfın özelliklerini ortaya koyar. O nedenle, sınıf değişkenlerine, çoğunlukla, özelik (`property`) denilir. Benzer olarak bir sınıf içinde tanımlı fonksiyonlar o sınıfın tavır, davranış ve eylemlerini ortaya koyar. Bu ayrımı gözetmek için olsa gerek, OO Programlama dillerinde fonksiyonlara `metot` denir. Bundan sonra, biz de bu geleneğe uyarak, bir sınıfta tanımlanan değişken yerine `özelik`, fonksiyon yerine `metot` diyeceğiz. Ancak, özellikle alışkanlıklarımıza dayanan vurgu gerektiğinde değişken ve fonksiyon adlarını kullanmaktan da çekinmeyeceğiz. Okurun, `değişken ile özeliğin` ve `fonksiyon ile metodun` eş anlamlı olduğunu daima anımsayacağını umuyoruz. Değişkenlere neden başka ad verdiğimizi birazdan anlamış olacağız.

Fonksiyon kavramına geçmeden önce, uzaylı dostumuzun bir uyarısı daha var. Onu dinlesek iyi olur.

Anadilimizi öğrenirken bile sistematik bir yöntem izlemedik. Bazı kalıpları (gramer) çevremizden duyduğumuz gibi kullanmaya başladık. Ana dilimizi kullanmayı öğrendikten çok sonra onun gramerini öğrendik. Programlama dillerinin öğrenimi de çoğunlukla ona benzer. Başlangıçta bir çok sözdizimini (syntax), nedenini sormadan,

olduğu gibi kullanırız. Sonra onun nedenini algılamamız daha kolay olur.

Static Öğe

Şimdiye dek bir çok kez kullandığımız `Main()` metodunu `static` anahtar sözcüğü ile niteledik. Başka metot ve değişkenler (özelik) için de aynı nitelemeyi kullandık. Bir sınıfta tanımlı bir değişken veya metodu `static` diye nitelemenin OO programlamada özel bir işlevi vardır. Genel kural olarak, bir sınıfta tanımlı değişkenler ve fonksiyonlar, ancak o sınıfa ait bir nesne (object) yaratıldıktan sonra o nesne içinde işlevsellik kazanırlar. Bunun istisnai hali değişkenin veya metodun önüne `static` nitelemesi konulduğunda oluşur. Başka bir deyişle şunu söyleyebiliriz:

İpucu

`static` sözcüğü ile nitelenen değişken veya metot, ait olduğu bir nesne (object) yaratılmadan kullanılabilir.

`static` sözcüğünün işlevini sınıf konusunu işlerken daha ayrıntılı açıklayacağımızı, ama o zamana kadar onu gerektiği yerde kullanmak zorunda olduğumuzu unutmayınız.

Metot Kavramı

`Metot` kavramı, matematik derslerinden bildiğimiz fonksiyon tanımından başka bir şey değildir. Matematikte

```
f : X --> Y  , {x -->y}
```

simgelerinin ifade ettiği anlam şudur: Fonksiyonun adı `f` dir. `f` fonksiyonu tanım kümesi denilen `X` kümesinden, değer kümesi denilen `Y` kümesi içine tanımlıdır. `f` fonksiyonu `X` kümesinden aldığı her `x` öğesini, fonksiyon kuralı denilen bir dönüşüm yöntemiyle `Y` kümesi içindeki `y` öğesine götürür (eşler). `X` öğesine fonksiyonun değişkeni, `y` öğesine `x` öğesinin `f` altındaki dönüşümü (resmi) diyoruz. Her `x` değişkenine bir ve yalnız bir `y` öğesi karşılık gelir. Başka bir deyişle, bir değişken birbirinden farklı iki öğeye eşlenemez. Buradan şunu hemen anlıyoruz. Her fonksiyonun bir tanım kümesi `( X )`, bir değer kümesi `( Y )` ve bir dönüşüm kuralı `{x --> y}` vardır.

Matematikten anımsadığımız bu kavramı, şimdi, bilgisayar diliyle ifade edelim. Açıklamayı basit bir örnek üzerinde yaparsak konu daha kolay anlaşılacaktır. Aşağıdaki fonksiyon bir tamsayının karesini hesaplar.

```
int KaresiniBul(short x)
{
    int y ;
    y = x * x ;
    return y ;
}
```

Bu tanımda

```
int KaresiniBul(short x)
```

satırına `fonksiyonun başlığı` diyeceğiz. Fonksiyon başlığı, fonksiyon hakkında bize gerekli başlıca bilgileri verir. Bu nedenle, bazı kaynaklar ona fonksiyonun mührü (signature) derler. Biz fonksiyon başlığı terimini tercih edeceğiz. Önce başlığın bize verdiği bilgilere bakalım:

Fonksiyonun adı `'KaresiniBul'` bitişik sözcükleridir.

Fonksiyonun adının sağına yazılan `(short x)` ifadesi, fonksiyonun tanım kümesinin `short` veri tipi olduğunu ve değişkeninin `x` simgesiyle gösterildiğini belirtir.

Fonksiyonun adının soluna yazılan `int` veri tipi fonksiyonun değer bölgesidir.

Başlıktaki satırdan sonra gelen { } bloku fonksiyona işlevini yaptıran deyimleri içeren bloktur. Bu bloku `fonksiyon bloku` veya `fonksiyon gövdesi` diye adlandıracağız.

{ } gövdesi içinde `y = x*x` çarpma ve atama işlemleri fonksiyonun yapacağı iştir. `x*x` çarpımının sonucu `y` değişkenine atanmış ve `y` değeri fonksiyon-değeri olarak belirlenmiştir. *[x öğesinin eşleneceği y öğesini belirleyen fonksiyon kuralı]*. Fonksiyonun alacağı değeri `return` anahtar sözcüğü ile belirtiyoruz:

```
return y;
```

Parametreler ve yerel değişkenler

Bilgisayar programlama dillerinde, fonksiyonun başlığındaki değişkene `parametre` denilir. Bunun nedenini açıklamak için, yukarıdaki { } gövdesine bakalım.

Fonksiyonun { } gövdesinde `int` tipinden bir `y` değişkeni tanımlanmıştır. Dolayısıyla bu fonksiyon iki farklı nitelikte değişken kullanmaktadır. Başlıktaki `(short x)` değişkeni ve fonksiyon gövdesi içindeki `int y` değişkeni.

Bu iki değişkeni birbirlerinden ayırt etmek için,

fonksiyon başlığındaki x değişkenine `parametre` ve

{ } gövdesi içindeki `y` değişkenine fonksiyonun `yerel değişkeni` denilir.

İpucu

Bilgisayar programlarında fonksiyona, matematikteki gibi tek harfli ad takmak yerine, genellikle, onun işlevini ima eden sözcükler takılır. Kaynak programı okuyan kişinin fonksiyonun işlevini hemen kavraması için bu iyi bir gelenektir. Bunu alışkanlık edininiz. Ancak, derleyici açısından fonksiyona takılan ad hiç önemli değildir. Tek harfli ad olsa da birden çok sözcükten oluşan uzun ad verilse de sistem ona işlevini eksiksiz gördürecektir.

Bilgisayar programlarında fonksiyonların parametrelerinin her biri bir veri tipi üzerinde tanımlıdır. Bu veri tip(ler)i onun tanım kümesidir. Her fonksiyon parametreleri ve yerel değişkenleri üzerinde, önceden belirlenen bazı işlemleri yapar ve sonunda bir değer verir (dönüşüm). Bu değer bir veri tipi içindedir. O veri tipi, fonksiyonun değer kümesidir.

C# dilinde her şeyin bir sınıf içinde tanımlanması gerektiğini söylemiştik. Öyleyse, yukarıdaki fonksiyon tanımını bir sınıf içine taşımalıyız. Fonksiyonumuzu istediğimiz bir sınıf içinde tanımlayabileceğimizi ileride göreceğiz. Ama, şimdi işleri kolaylaştırmak için onu, `Main()` fonksiyonunu da içeren bir sınıf içine taşıyalım.

Metot01.cs

```
using System;

namespace Metotlar
{
    class Metot01
    {
        static int n;

        static void Main(string[] args)
        {
            n = KaresiniBul(9);
            Console.WriteLine(n);
        }

        static int KaresiniBul(short x)
        {
            int y;
            y = x * x;
            return y;
        }
    }
}
```

Main() metodu

C# dilinde bir programın bütün işlevini yaptıran ana metottur. Programın kullanacağı bütün deyimleri tek başına içerebileceği gibi, modüler yapıda yazılan programlarda öteki sınıflarda tanımlanan metotları ve değişkenleri çağırır ve belirlenen sırayla işleme sokar. Her programda ana metot olma işlevini üstlenen bir ve yalnızca bir tane `Main()` metodu vardır. Bir programda başka Main metotları olsa bile, onlar bu programda ana metot olma sorumluluğunu taşıyamazlar.

`Main()` bir metot olduğuna göre, tanımı genel sözdizimi kurallarına uyar. Çoğunlukla başlığını

```
static void Main(string[] args)
```

biçiminde yazarız. Bunun nedenini ilerleyen bölümlerde daha iyi anlayacaksınız. Şimdilik şunları söylemekle yetinelim: Yukarıdaki `Main()` metodunun,

`args` adını taşıyan parametresi `string` tipi bir array'dir.

Değer kümesi `void` kümesidir. Bu küme matematikteki `boş küme` gibidir. Hiç bir öğesi yoktur. Dolayısıyla, bu örnekte `Main()` metodu bir değer almıyor, yalnızca başka bir metodu çağırıp onun işlevini yapmasını sağlıyor.

Başlıktaki `static` anahtar sözcüğünü mutlaka kullanmalıyız. Onun ne iş yaptığını ileride göreceğiz.

Yukarıdaki örnekte, `Metot01` adlı sınıf içinde,

```
static int n;
```

değişkeni tanımlıdır. Sınıfa ait değişkenlere özelik diyeceğimizi söylemiştik. Dolayısıyla, `n` değişkeni `Metot01` sınıfının bir özeliğidir. Ayrıca bu sınıf içinde `Main()` metodu ile `KaresiniBul()` metodu tanımlıdır. `Main()` metodunun gövdesinde yazılan

```
KaresiniBul(9);
```

deyimine, *fonksiyonu parametre değeriyle çağırmak* denir. Fonksiyon çağrılırken onun `x` parametre adı yerine, o parametreye verilen 9 değeri ile çağrılmaktadır. Başka bir deyişle, fonksiyon çağrılırken `x*x` soyut işlemi yerine `9*9` somut işlemini yapması istenmektedir. Bu olgu, matematikte `f(x) = x^2` fonksiyonu tanımlı iken `f(9) = 9^2 = 81` işlemini yapmak gibidir.

`Metot02` sınıfı içinde tanımlanan `n` değişkeni, `KaresiniBul()` metodu ve `Main()` metodu `static` anahtar sözcüğü ile nitelenmişlerdir. Bunun nedenini bu bölümün başında açıklamıştık. Ayrıca, anımsayacaksınız, bu bölümün başında sınıf içinde tanımlanan değişkenlere '`özelik`' diyeceğiz demiştik.

İpucu

Böylece, bir sınıf içinde olabilecek üç farklı nitelikteki değişkeni ayrı ayrı gruplamış oluyoruz:

Fonksiyonun yerel değişkeni,

fonksiyonun parametresi ve

sınıfın özeliği.

Yukarıdaki basit örnekte görüldüğü gibi, bu değişken gruplarının kullanılışları ve işlevleri birbirlerinden farklıdır.

Ayrıca bir fonksiyonun hiç parametresi veya yerel değişkeni olmayabileceği gibi, birden çok parametresi veya yerel değişkeni olabilir.

Benzer olarak, bir sınıfın hiç özeliği veya metodu olmayabileceği gibi, birden çok özeliği ve metodu olabilir. Bunları örnekler üzerinde göreceğiz.

Aşağıdaki program bir metin içindeki küçük harf, büyük harf, rakam ve `alfasayısal` (harf veya rakam) olmayan karakterlerin sayılarını bulur. Her bir sayımı yapmak için, `KarakterleriSay` sınıfı içinde dört ayrı metot tanımlanmıştır. Metotların `Main()` tarafından çağrılabilmesi için herbiri `static` nitelemesini almıştır.

KarakterleriSay.cs

```
using System;

namespace Metotlar
{
    class KarakterleriSay
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Bir satır yazıp Enter'e basınız: ");
            string s = Console.ReadLine();
            Console.WriteLine("Küçük Harf Sayısı : " + KüçükHarfSay(s));
```

```
            Console.WriteLine("Büyük Harf Sayısı : " + BüyükHarfSay(s));
            Console.WriteLine("Rakam Sayısı : " + RakamSay(s));
            Console.WriteLine("Alfasayısal Olmayan Karakter Sayısı :" +
AlfasayısalDeğilseSay(s));
        }


        static int KüçükHarfSay(string str)
        {
            int küçükHarfSayısı = 0;
            foreach (char harf in str)
                if (Char.IsLower(harf)) küçükHarfSayısı++;
            return küçükHarfSayısı;
        }

        static int BüyükHarfSay(string str)
        {
            int büyükHarfSayısı = 0;
            foreach (char harf in str)
                if (Char.IsUpper(harf)) büyükHarfSayısı++;
            return büyükHarfSayısı;
        }

        static int RakamSay(string str)
        {
            int rakamSayısı = 0;
            foreach (char harf in str)
                if (Char.IsDigit(harf)) rakamSayısı++;
            return rakamSayısı;
        }

        static int AlfasayısalDeğilseSay(string str)
        {
            int alfaSayısalDeğilSayısı = 0;
            foreach (char harf in str)
                if (!Char.IsLetterOrDigit(harf)) alfaSayısalDeğilSayısı+
+;
            return alfaSayısalDeğilSayısı;
        }


    }
}
```

Çıktı

```
Bir satır yazıp Enter'e basınız:
C# dersine başlayalı 3 gün oldu. Şimdi program yazabiliyorum.
Küçük Harf Sayısı : 47
Büyük Harf Sayısı : 2
Rakam Sayısı : 1
Alfasayısal Olmayan Karakter Sayısı :11
Devam etmek için bir tuşa basın . . .
```

`static` nitelemesinin işlevini daha iyi açıklamak için Birkaç örnek program daha inceleyeceğiz. Aşağıdaki programı derlemeyi deneyiniz.

Tarih01.cs

```
using System;

namespace Metotlar
{
    class Tarih
    {
            static void Main(string[] args)
            {
                TarihYaz();
            }


            void TarihYaz()
            {
                DateTime gününTarihi = DateTime.Now;
                Console.WriteLine(gününTarihi.ToString("dd/MM/yyyy"));
            }
        }
    }
```

Derleyici size şu hata mesajını iletecektir:

```
Error   1   An object reference is required for the non-static field, method, or property 'Metotlar.Tarih.TarihYaz()'
    C:\vsProjects\Metotlar\Metotlar\Metot01.cs   9   17   Metotlar
```

Bu iletinin bize söylediği şey şudur: 9-uncu satırdaki `static` olmayan `TarihYaz()` metodu ancak bir nesne içinde kullanılabilir.

Şimdi bir nesne yaratmadan, `static` nitelemesiyle derleyicinin itirazını kaldırabileceğimizi, aşağıdaki program gösterecektir. Bunun için `TarihYaz()` metodunu `static` olarak nitelemek yetecektir.

Tarih02.cs

```
using System;

namespace Metotlar
{
    class Tarih
    {
            static void Main(string[] args)
            {
                TarihYaz();
            }


            static void TarihYaz()
            {
                DateTime gününTarihi = DateTime.Now;
                Console.WriteLine(gününTarihi.ToString("dd/MM/yyyy"));
            }
        }
```

```
    }
```

Çıktı

```
10/08/2008
```

Nesne yaratma

Şimdi de `static` nitelemesini kullanmamak için `Tarih` sınıfından bir nesne yaratalım. Sınıfları incelerken bu konuyu daha ayrıntılı ele almıştık. Burada şu kadarını tekrar anımsayalım: `Tarih` sınıfı tanımlı iken, `Main()` metodu içinde

```
Tarih t = new Tarih();
```

deyimi `Tarih` sınıfının `t` adlı bir nesnesini yaratır. Bu nesne içindeki `TarihYaz()` metodunu çağırmak (nesne içindeki metoda erişmek) için

```
t.TarihYaz();
```

deyimi yeterlidir. Aşağıdaki programı derleyip koşturunuz; önceki program ile aynı çıktıyı verdiğini göreceksiniz.

Tarih03.cs

```
using System;

namespace Metotlar
{
    class Tarih
    {
            static void Main(string[] args)
            {
                Tarih t = new Tarih();
                t.TarihYaz();
            }


            void TarihYaz()
            {
                DateTime gününTarihi = DateTime.Now;
                Console.WriteLine(gününTarihi.ToString("dd/MM/yyyy"));
            }
        }
    }
```

Çıktı

```
10.07.2008

Devam etmek için bir tuşa basın . . .
```

Hazır metotlar

C# derleyicisi çok zengin bir kütüphaneye sahiptir. Bu kütüphanede çok sayıda namespace, her namespace içinde sınıflar ve sınıflar içinde pek çok işi görecek metotlar vardır. Buna rağmen, kullanıcı gerekseme duyduğunda kendi namespace'lerini, sınıflarını ve metotlarını serbestçe yaratabilmektedir. Yukarıda yaptıklarımız, kullanıcının yaratabileceği sınıf ve metotların basit örnekleridir.

Şimdiye kadar kullandığımız `Write()`, `WriteLine`, `Read()`, `ReadLine()` metotları `Console` sınıfına aittir. `isLower()`, `isUpper()`, `isDigit()`, `IsLetterOrDigit()` metotları `Char` sınıfına aittir. İlerleyen bölümlerde C# kütüphanesindeki başka sınıf ve metotları kullanmayı öğreneceğiz. Bunlar hakkında daha ayrıntılı bilgiyi `msdn` web sitesinden her zaman öğrenebilirsiniz. Ama başlangıçta kütüphaneyi ezberlemeye kalkmayın. Sizden beklenen, gerektiğinde o kütüphaneyi nasıl kullanacağınızı ve aradığınızı nasıl bulacağınızı öğrenmenizdir.

System.Math sınıfı

Matematik fonksionları her dilde çok önem taşır. Onlar olmadan program yazmak hemen hemen olanaksızdır. C# dili matematik metotlarını `System.Math` sınıfı içinde toplamıştır. Math sınıfı mühürlüdür (sealed). Dolayısıyla, ileride göreceğimiz gibi, Math sınıfının kalıtımı (inherited) olamaz. Math sınıfının bütün öğeleri static'dir. O nedenle, Math sınıfının bir nesnesi (object) yaratılamaz. Böyle olduğu için, Math sınıfının metotlarını Math.Sqrt() biçiminde sınıf adıyla birlikte kullanacağız.

Aşağıdaki program o metotları ve çağrılma yöntemlerini göstermektedir.

MathMetotları.cs

```
using System;

namespace Metotlar
{
    public class MathMetotları
    {
        static void Main(string[] args)
        {
            double x,y;
            string sx, sy ;
            Console.WriteLine("Lütfen birinci sayıyı giriniz : ");
            sx = Console.ReadLine();
            x = Double.Parse(sx);

            Console.WriteLine("Lütfen ikinci sayıyı giriniz : ");
            sy = Console.ReadLine();
            y = Double.Parse(sy);

            Console.WriteLine();

            Console.WriteLine("Başlıca Math Metotları :");
            Console.WriteLine("Abs({0}) = {1}", x, Math.Abs (x));
            Console.WriteLine("Ceiling({0}) = {1}", x, Math.Ceiling(x));
            Console.WriteLine("Exp({0}) = {1}", x, Math.Exp(x));
            Console.WriteLine("Floor({0}) = {1}", x, Math.Floor(x));
            Console.WriteLine("IEEERemainder({0},{1}) = {2}", x, y,
Math.IEEERemainder(x,y));
```

```
            Console.WriteLine();
            Console.WriteLine("Log({0}) = {1}", x, Math.Log(x));
            Console.WriteLine("Log10({0}) = {1}", x, Math.Log10(x));
            Console.WriteLine("Max({0}) = {1}", x, Math.Max(x, y));
            Console.WriteLine("Min({0},{1}) = {2}", x, y, Math.Min(x,y));
            Console.WriteLine("Pow({0})  = {1}", x, Math.Pow(x, y));
            Console.WriteLine("Round({0}) = {1}", x, Math.Round(x));
            Console.WriteLine("Sign({0})  = {1}", x, Math.Sign(x));
            Console.WriteLine("Sqrt({0})  = {1}", x, Math.Sqrt(x));
            Console.WriteLine();
            Console.WriteLine("Cos({0})   = {1}", x, Math.Cos(x));
            Console.WriteLine("Sin({0})   = {1}", x, Math.Sin(x));
            Console.WriteLine("TAn({0})   = {1}", x, Math.Tan(x));
            Console.WriteLine("Cotan({0}) = {1} ", x, 1 / Math.Tan(x));
            Console.WriteLine();
            Console.WriteLine("Acos({0})  = {1}", x, Math.Acos(x));
            Console.WriteLine("Asin({0})  = {1} ", x, Math.Asin(x));
            Console.WriteLine("Atan({0})  = {1} ", x, Math.Atan(x));
            Console.WriteLine("Atan2({0},{1}) = {2}", x,y,
Math.Atan2(x,y));
            Console.WriteLine();
            Console.WriteLine("Cosh({0})  = {1} ", x, Math.Cosh(x));
            Console.WriteLine("Sinh({0})  = {1} ", x, Math.Sinh(x));
            Console.WriteLine("TAnh({0})  = {1} ", x, Math.Tanh(x));
            Console.WriteLine("Cotanh({0}) = {1} ", x, 1 / Math.Tanh(x));
        }
    }
}
```

Çıktı

```
Lütfen birinci sayıyı giriniz :
456
Lütfen ikinci sayıyı giriniz :
123
```

Başlıca Math Metotlarının kullanılışı:

Abs(456)	= 456
Ceiling(456)	= 456
Exp(456)	= 1,09215366597392E+198
Floor(456)	= 456
IEEERemainder(456,123)	= -36

Log(456)	= 6,12249280951439
Log10(456)	= 2,65896484266443
Max(456)	= 456
Min(456,123)	= 123
Pow(456)	= Infinity
Round(456)	= 456
Sign(456)	= 1
Sqrt(456)	= 21,3541565040626
Cos(456)	= -0,891991243357829
Sin(456)	= -0,45205267588297
TAn(456)	= 0,506790486172548
Cotan(456)	= 1,97320199823074
Acos(456)	= NaN
Asin(456)	= NaN
Atan(456)	= 1,56860334785422
Atan2(456,123)	= 1,30732978575998
Cosh(456)	= 5,4607683298696E+197
Sinh(456)	= 5,4607683298696E+197
TAnh(456)	= 1
Cotanh(456)	= 1

Özyineli (Recursive) Metotlar

Recursive (özyineli) metot kendi kendisini çağıran metottur. Bilgisayar programcılığında zor problemleri çözen önemli bir algoritmadır.

Fibonacci sayıları

Fibonacci sayıları doğanın çoğalma formülü diye bilinir. Biyolojide, tıpta ve mühendislikte önemli uygulamaları vardır. *Fibonacci* sayılarının ilk 10 tanesini hesaplayan aşağıdaki program recursive bir metot kullanmaktadır.

FibonacciSayıları.cs

```
using System;

namespace Methods
{
    class FibonacciSayıları
    {
        static void Main(string[] args)
        {
            for(int i=1; i<11;i++)
            Console.Write(Fibonacci(i) + "   ");
            Console.WriteLine();
        }

        static int Fibonacci(int n)
        {
            if (n < 2)
                return n;
            else
                return Fibonacci(n - 1) + Fibonacci(n - 2);
        }
    }
}
```

Çıktı

```
1  1  2  3  5  8  13  21  34  55
Devam etmek için bir tuşa basın . . .
```

Recursive metotlar istenen işi yapmak için kendi kendilerini defalarca çağırırlar. Bunu daha iyi algılamak için, yukarıdaki `Fibonacci()` metodunun kaç kez çağrıldığını saydıralım. Bunun için bir sayaç değişkeni tanımlayıp her adımda 1 artmasını sağlamak yetecektir. Bunu yapan aşağıdaki program aynı zamanda Fibonacci sayılarının ne kadar hızla büyüdüğünü göstermektedir. Bunu canlıların çoğalmasına, örneğin bakterilerin çoğalmasına uygularsak Fibonacci sayılarının neden önemli olduğunu anlarız.

FibonacciSayıları02.cs

```
using System;

namespace Methods
{
    class FibonacciSayıları
```

```
    {
        static long sayaç = 0;
        static void Main(string[] args)
        {
            for (int i = 1; i <= 45; i++)
                Console.WriteLine("Fibonacci({0}) = {1:###,###,###,##0}
[{2:###,###,###,###} adım atıdı]", i, Fibonacci(i), sayaç);
        }

        static int Fibonacci(int n)
        {
            sayaç++;
            if (n < 2)
                return n;
            else
                return Fibonacci(n - 1) + Fibonacci(n - 2);
        }
    }
}
```

Çıktı

Fibonacci(n)	**Değeri**	**Kaç Adım**
Fibonacci(1)	= 1	[1 adım atıdı]
Fibonacci(2)	= 1	[4 adım atıdı]
Fibonacci(3)	= 2	[9 adım atıdı]
Fibonacci(4)	= 3	[18 adım atıdı]
Fibonacci(5)	= 5	[33 adım atıdı]
Fibonacci(10)	= 55	[452 adım atıdı]
Fibonacci(15)	= 610	[5.149 adım atıdı]
Fibonacci(20)	= 6.765	[57.290 adım atıdı]
Fibonacci(25)	= 75.025	[635.593 adım atıdı]
Fibonacci(30)	= 832.040	[7.049.122 adım atıdı]
Fibonacci(35)	= 9.227.465	[78.176.299 adım atıdı]
Fibonacci(40)	= 102.334.155	[866.988.830 adım atıdı]
Fibonacci(45)	= 1.134.903.170	[9.615.053.903 adım atıdı]

Faktöriyel hesabını da *recursive (özyineli)* metot ile yapabiliriz.

Aşağıdaki iki metodu inceleyiniz. Birincisi `n!` Sayısını döngü ile, ikincisi özyineli yöntemle buluyor.

```
        int faktoriyel = 1;

            for (int i = n; i >= 1; i--)
                faktoriyel *= i;

        private static long Faktoriyel(int n)
        {
            if (n <= 1)
                return 1;
            else
                return n * Faktoriyel(n - 1);
        }
```

```
using System;

namespace Metotlar
{
    public class Program01
    {
        static void Main(string[] args)
        {
            Console.WriteLine(WhileContinue());
            WhileBreak();
            WhileGoto();
        }
        static int WhileContinue()
        {
            int a = 0;

            while (a < 10)
            {
                a++;
                if (a == 5)
                {
                    a++;
                    continue;
                }
            }
            return a;

        }
        static void WhileBreak()
        {
            int a = 0;

            while (a < 10)
```

```
            {
                a++;
                if (a == 5)

                    break;
            }
            Console.WriteLine(a);
        }

        static void WhileGoto()
        {
            int a = 0;

            while (a < 40)
            {
                if (a == 27)
                    goto sil;
                a++;
            }
        sil:

            Console.WriteLine(a);
        }
    }
}
```

Çıktı

```
10

5

27
```

Kullanıcının Tanımladığı Metotlar

Kendi metodunuzu yaratmanız gerekebilir.

ToplamBul.cs

```
using System;

namespace Methods
{
    class ToplamBul
    {
        static double a = 12.34;
        static double b = 56.78;
        static void Main(string[] args)
        {
            Console.WriteLine(Topla(a,b));
        }

        static double Topla(double x, double y)
        {
            return x+y ;
        }
    }
```

```
}
```

Çıktı

```
69,12
```

Program yazarken, derleyici kütüphanesindeki hazır fonksiyonları kullanmamaya alışmak iyi bir alışkanlıktır. Onlar program yazmayı kolaylaştırdıkları gibi, daima doğru işlem yaptıklarına da güvenebiliriz. Ama zorunluluk olduğunda kendi fonksiyonlarımızı yaratabiliriz.

Aşağıdaki fonksiyon, int tipinden sayıların karekökünü hesaplamak için yazılmıştır. Karekökün tamsayı kısmını doğru hesaplar. Aynı sonucu `Math.Sqrt()` metoduna yaptırmak çok daha kolay ve güvenlidir.

KareKök.cs

```
using System;

namespace Methods
{
    class KareKök
    {
        static int n = 1234;
        static void Main(string[] args)
        {
            Console.WriteLine(Isqrt(n));
            Console.WriteLine(Math.Sqrt(n));
        }

        // int tipli sayının tamsayı karekökünü bulur
        public static int Isqrt(int num)
        {
            if (0 == num) { return 0; }  // Sıfıra bölmeyi önler
            int n = (num / 2) + 1;       // İlk tahmin
            int n1 = (n + (num / n)) / 2;
            while (n1 < n)
            {
                n = n1;
                n1 = (n + (num / n)) / 2;
            }
            return n;
        }
    }
```

Çıktı

```
35
35,1283361405006
```

Bazan hazır fonksiyonlardan yararlanarak kendi fonksiyonumuzu yaratabiliriz.

```
using System;

namespace Methods
{
```

```
    class LogaritmaBul
    {
        static int n = 1234;
        static void Main(string[] args)
        {
            Console.WriteLine(Math.Log(n));
        }

        static double LogBul(double y)
        {
            return Math.Log(y);
        }
    }
}
```

Çıktı

```
7,11801620446533
```

Uygulamalar

Matematik derslerinde verilen sayıların en küçük ortak katını (ekok) ve en büyük ortak bölenini (ebob) bulmak ilköğretimin önemli konularındandır. Şimdi bunu C# nasıl bulacağımızı gösterelim.

Ekok

```
/* a ve b ekok bulunacak sayılar */
  int ekokBul(int a,int b)
  {
    int n;
    for(n=1;;n++)
    {
        if(n%a == 0 && n%b == 0)
          return n;
    }
  }
```

```
using System;

namespace Metotlar
{
    public class MathMetotları
    {
        static int n, m, ekok;
        static string s;
        static void Main(string[] args)
        {
            Console.WriteLine("Lütfen ekok 'u bulunacak sayıların ilkini
giriniz:");
            s = Console.ReadLine();
            n = Int32.Parse(s);
```

```
            Console.WriteLine("Lütfen ekok 'u bulunacak sayıların
ikincisini giriniz:");
            s = Console.ReadLine();
            m = Int32.Parse(s);
            Console.WriteLine(ekokBul(n, m));
        }

        /* a ve b ekok'u bulunacak sayılar */
        static int ekokBul(int a, int b)
        {
            int n;
            for (n = 1; ; n++)
            {
                if (n % a == 0 && n % b == 0)
                    return n;
            }
        }
    }
}
```

Çıktı

```
Lütfen ekok 'u bulunacak sayıların ilkini giriniz:
35
Lütfen ekok 'u bulunacak sayıların ikincisini giriniz:
15
105
```

Aşağıdaki program `ekok` bulmak için *recursive* metot kullanmaktadır.

EkokBul02.cs

```
using System;

namespace Metotlar
{
    public class EkokBul02
    {
        static int x, y, ekok;
        static string s;
        static void Main(string[] args)
        {
            Console.WriteLine("Lütfen ekok 'u bulunacak sayıların ilkini
giriniz:");
            s = Console.ReadLine();
            x = Int32.Parse(s);
            Console.WriteLine("Lütfen ekok 'u bulunacak sayıların
ikincisini giriniz:");
            s = Console.ReadLine();
           y = Int32.Parse(s);
            Console.WriteLine(EkokBul(x, y));
        }
```

```
        static int EkokBul(int x, int y)
        {
            int prod;
            if (y % x == 0)
                return y;
            else
            {
                prod = x * y;
                while (x != y)
                {
                    if (x > y)
                        x = x - y;
                    else
                        y = y - x;
                }
                return EkokBul(y, prod / x);
            }
        }
    }
}
```

Çıktı

```
Lütfen ekok 'u bulunacak sayıların ilkini giriniz:
75
Lütfen ekok 'u bulunacak sayıların ikincisini giriniz:
45
225
```

Aşağıdaki program iki tamsayının en büyük ortak bölenini (EBOB) bulur.

```
using System;

public class EbobHesabı
{
    static int EbobBul(int a, int b)
    {
        int kalan;

        while (b != 0)
        {
            kalan = a % b;
            a = b;
            b = kalan;
        }

        return a;
    }

    static int Main(string[] args)
    {
        int x, y;

        Console.WriteLine("Ebob'i bulunacak sayıları giriniz : ");
```

```
        Console.Write("Sayı 1: ");
        x = int.Parse(Console.ReadLine());
        Console.Write("Sayı 2: ");
        y = int.Parse(Console.ReadLine());

        Console.WriteLine();
        Console.WriteLine("{0} ile {1} sayılarının ebob'ni: {2}", x, y,
EbobBul(x, y));

        return 0;
    }
}
```

Çıktı

```
Ebob'i bulunacak sayıları giriniz :
Sayı 1: 124
Sayı 2: 68

124 ile 68 sayılarının ebob'ni: 4
```

Aşağıdaki program ebob bulmak için *recursive* metot kullanmaktadır.

EbobHesabı02.cs

```
using System;

public class EbobHesabı
{
    static int EbobBul(int a, int b)
    {
        int c;
        while (a % b != 0)
        {
            c = a % b;
            a = b;
            b = c;
        }
        return b;
    }

    static int Main(string[] args)
    {
        int x, y;

        Console.WriteLine("Ebob'i bulunacak sayıları giriniz : ");
        Console.Write("Sayı 1: ");
        x = int.Parse(Console.ReadLine());
        Console.Write("Sayı 2: ");
        y = int.Parse(Console.ReadLine());

        Console.WriteLine();
        Console.WriteLine("{0} ile {1} sayılarının ebob'ni: {2}", x, y,
EbobBul(x, y));
```

```
        return 0;
    }
}
```

Çıktı

```
Ebob'i bulunacak sayıları giriniz :
Sayı 1: 128
Sayı 2: 64

128 ile 64 sayılarının ebob'ni: 64
```

İkiden çok sayının `ebob` 'ni bulmak için, `EbobBul()` metodunu ve birleşme (associative) özeliğini kullanmak yetecektir. Örneğin, üç sayının `ebob` 'ni bulmak için,

```
EBOB(a, b, c) = EbobBul(EbobBul(a,b), c)
```

birleşme özeliğini kullanabiliriz:

EBOB.cs

```
using System;

public class EbobHesabı
{
    static int EbobBul(int a, int b)
    {
        int c;
        while (a % b != 0)
        {
            c = a % b;
            a = b;
            b = c;
        }
        return b;
    }

    static int Main(string[] args)
    {
        int x, y, z;

        Console.WriteLine("Ebob'i bulunacak üç sayıyı giriniz : ");
        Console.Write("Sayı 1: ");
        x = int.Parse(Console.ReadLine());

        Console.Write("Sayı 2: ");
        y = int.Parse(Console.ReadLine());

        Console.Write("Sayı 3: ");
        z = int.Parse(Console.ReadLine());
```

```
        Console.WriteLine();
        Console.WriteLine("{0}, {1} ve {2} sayıları için ebob : {3}", x,
y, z, EbobBul(EbobBul(x, y), z));

        return 0;
    }
}
```

Çıktı

```
Ebob'i bulunacak üç sayıyı giriniz :
Sayı 1: 24
Sayı 2: 48
Sayı 3: 64

24, 48 ve 64 sayıları için ebob : 8
```

Alıştırmalar

`string`'den tamsayıya dönüşüm:

```
using System;

namespace Metotlar
{
    class Dönüşümler
    {
        static void Main(string[] args)
        {
            string s = "1234";
            int  n = Int32.Parse(s); Console.WriteLine(n);
            int  m = int.Parse(s);   Console.WriteLine(m);
            int  p = short.Parse(s); Console.WriteLine(p);
            long q = long.Parse(s);  Console.WriteLine(q);
            Console.WriteLine();
        }
    }
}
```

Bölüm 13

Yapı (struct)

Yapı Nedir?

C# dilinde yapılar farklı veri tiplerinden oluşan bir karma yapıdır. Sınıflara benzerler; onlar gibi tanımlanır, nesneleri onlar gibi yaratılır. Alanlar (field), metotlar, numaratörler (indexer) ve hatta başka yapıları öğe olarak içerebilirler. Ancak sınıf ile yapı arasında çok önemli bir ayrım vardır. Sınıfın nesneleri `heap` 'te tutulurken, yapı 'nın nesneleri `stack` 'ta tutulur. Böyle oluşu bazan iyidir, ama bazan da yapının öğe sayısına bağlı olarak elverişsiz olabilir. Neden öyle olduğunu birazdan anlayacağız.

C# dilinde değişkenlerin `değer (value)` ve `referans (reference)` tipler olmak üzere ikiye ayrıldığını; ana bellek içinde değer tiplerinin `stack`'ta, referans tiplerinin `heap`'te tutulduklarını biliyoruz.

`String` dışında bütün temel veri tipleri (built-in-types) `değer` tipidir; dolayısıyla `stack`'ta tutulurlar. Nesneler ve string `referans` tipidir, `heap`'te tutulurlar. `Stack`'ta tutulan öğelerin işi bitince kendiliğinden bellekten silinirler. Referans tiplerin işi bitince, çöp toplayıcı (garbage collector) onları toplayıp siler ve boşalan bellek bölgesini heap'e katar.

Java dilinde `struct` yoktur; programcı her şeyi sınıflarla yapar. C# dili sınıfa ek olarak `struct` yapısını ortaya koymuştur. Sınıftan yaratılan nesneler heap'te yer alıyordu. Bunun aksine, nesneleri stack'ta yer alacak bir alternatif oluşturmuştur.

Programcı stack'ta yer alan iki yapı kurabilir.

1. struct
2. enum

`enum` yapısını ileride göreceğiz. Bu bölümde `struct`'u inceleyeceğiz.

Yapı Bildirimi

Yapı bildirimi sınıf bildirimi gibi yapılır; class yerine struct yazılır:

```
<belirteçler> struct  <struct_adı>
{
//Struct 'un öğeleri
}
```

Örnek:

```
public struct DenekYapı
{
    public int a;
    public int b;
}
```

Bir yapı `private, public, internal` ya da `public` erişim belirteçlerinden birini alabilir.

New Operatörü ile Yapı Nesnesi Yaratma

Yapıya ait bir nesne yaratma aynen sınıflarda yapıldığı gibidir:

```
DenekYapı denek = new DenekYapı();
```

deyimi `stack` içinde `denek` adlı bir nesne yaratır. Yapının öğelerine erişim sınıflar için yapıldığı gibidir:

```
denek.a = 15;
denek.b = 20;
```

deyimleri yapı'ya ait denek nesnesinin öğelerine değer atamaktadır.
Sınıflarda olmadığı halde yapılarda var olan özeliklerden birisi de şudur:

```
DenekYapı d;
```

deyimi DenekYapı'ya ait d adlı bir nesne yaratır. Bu olgu temel veri tiplerinde yaptığımız

```
int n;
```

değişken bildirimine benziyor.
Aşağıdaki programı çözümleyelim.

Yapı01.cs

```
using System;

struct Kimlik
{
    public string ad;
    public int   yaş;


    static void Main(string[] args)
    {

        Kimlik y = new Kimlik();
        y.ad = "Dilek";
        Console.WriteLine(y.ad);
        Console.WriteLine(y.yaş);
    }
}
```

Çıktı

```
Dilek
0
```

`Kimlik` adlı struct içinde iki tane *alan (field)* tanımlıdır: `ad` ve `yaş`. Main() metodu içinde `new` opratörü ile Kimlik yapısının `y` adlı bir nesnesi yaratıldı. Bu nesne stack içindedir. `y` nesnesinin `ad` alanına "Dilek" adı atandı, ama `yaş` alanına bir değer atanmadığı için, öndeğer (default value) olarak 0 değerini tutuyor. Program derlenir, ama derleyici `yaş` alanına değer atanmadığı için öndeğer olarak 0 tuttuğu mesajını iletir. Zaten böyle olduğunu bu iki alanın değerlerini konsola yazdıran son iki deyimin çıktısından görüyoruz.

Bu programda başka bir özelik göze çarpıyor. Şimdiye kadar `Main()` metodunu daima bir sınıf (class) içinde tanımladık. Bu program, `Main()` metodunun bir yapı (struct) içinde de tanımlanabileceğini gösteriyor. Zaten, yapı ile sınıf bir çok konuda benzer işlevlere sahiptirler.

Sınıf İçinde Yapı Bildirimi

İstersek, yapıları sınıf içinde de tanımlayabiliriz. Aşağıdaki program onun yapılabilirliğini gösteriyor.

Yapı02.cs

```
using System;
class Yapılar
{
    struct Kimlik
    {
        public string ad;
        public int yaş;
    }

    static void Main(string[] args)
    {
        Kimlik y = new Kimlik();
        y.yaş = 19;
        Console.WriteLine(y.ad);
        Console.WriteLine(y.yaş);
    }
}
```

Çıktı

```
19
```

`Yapılar` adlı sınıfın içine `Kimlik` adlı yapı (struct) ile `Main()` metodu yerleşmiştir. Önceki programdan farklı olarak bu kez `ad` alanına değer atanmamış, `yaş` alanına 19 değeri atanmıştır. Çıktıdan görüldüğü gibi, konsola, string tipi olan `ad` değişkeninin öndeğeri olan "" boş string yazılmıştır.

Uyarı

Bu programda yapının alanlarına verilen public nitelemesini kaldırırsanız, derleyici onlara erişemediğini belirten şu iletiyi gönderecektir:

```
Error  1   'Yapılar.Kimlik.yaş' is inaccessible due to its protection level ...
```

New Operatörü Kullanmadan Yapı Nesnesi Yaratma

Şimdi new operatörünü kullanmadan, temel veri tiplerinde yaptığımız gibi yapı tipinden değişken tanımlayalım. Bunun için yukarıdaki programı aşağıdaki gibi değiştirelim.

Yapı03.cs

```
using System;
class Yapılar
{
    struct Kimlik
    {
        public string ad;
        public int yaş;
    }

    static void Main(string[] args)
    {
        Kimlik k;
        k.ad = "Yasemin";
        Console.WriteLine(k.ad);
        Console.WriteLine(k.yaş);
    }
}
```

Programı derlemek istersek şu hata mesajını alırız:

```
Error  1  Use of possibly unassigned field 'yaş' ...
```

Bu mesajdan anlıyoruz ki, yapıyı temel veri tipi gibi kullanarak değişken tanımlarsak, onun bütün öğelerine değer atamadan değişkeni kullanamayız. Gerçekten, ikinci değişkene de değer atarsak, programın derlendiğini görebiliriz. Bunun için yukarıdaki programda son satırdan önce

```
k.yaş = 21;
```

deyimini eklemeniz yetecektir.

Sınıf değişkenlerine bildirim anında ilk değer atanabiliyordu. Ama yapı içinde bildirimi yapılan alanlara, ilk değer atanamaz. Bu gerçeği görmek için, ilk programımızda değişkenlere değer atayıp derlemeyi deneyelim.

Yapı04.cs

```
using System;

struct Kimlik
{
    public string ad = "Zeki Alasya";
    public int yaş = 70;


    static void Main(string[] args)
    {
        Kimlik y = new Kimlik();
        Console.WriteLine(y.ad);
        Console.WriteLine(y.yaş);
    }
}
```

Derleyicimiz şu hata iletisini gönderecektir.

```
Error  1   'Kimlik.ad': cannot have instance field initializers in structs ...
```

Burada karşılaştığımız engeli aşmanın bir yolu var. Daha önce kullandığımız static nitelemesini kullanmak. Eğer yapının değişkenlerine (alan, field) `static` nitelemesini verirsek, engel ortadan kalkar. Tabii, bu durumda `new` operatörü ile yapıya ait bir nesne yaratmaya gerek kalmaz.

Yapı05.cs

```
using System;

struct Kimlik
{
    static string ad = "Zeki Alasya";
    static int yaş = 70;


    static void Main(string[] args)
    {
        Console.WriteLine(Kimlik.ad);
        Console.WriteLine(Kimlik.yaş);
    }
}
```

Bu programda `Kimlik` yapısının `static` değişkenlerine erişmek için, bir nesne adı değil, yapının adının kullanıldığına dikkat ediniz. Aynı olgunun sınıflar için de olduğunu görmüştük.

İlk programımızda aynı yapı içinde `Main()` metodunun bildirimini yaptık. İkinci programda hem yapıyı hem `Main()` metodunu bir sınıfın içine koyduk. Şimdi `Main()` metodunu yapı dışındaki bir sınıfın içine alıp sonucu görelim.

Yapı06.cs

```
using System;
struct Hesap
{
    public int a;
    public int b;
}
class Uygulama
{
    public static void Main()
    {
        Hesap yy = new Hesap();
        yy.a = 15;
        yy.b = 25;
        int toplam = yy.a + yy.b;
        Console.WriteLine("Toplam = {0}", toplam);
    }
}
```

Bu programda `Hesap` adlı yapı ile `Uygulama` adlı sınıf birbirinin dışındadırlar. `Uygulama` sınıfı içinde tanımlanan `Main()` metodu `Hesap` yapısı içindeki alanlara erişebilmektedir.

Yukarıda söylediğimiz gibi, `Hesap` yapısının alanlarından `public` nitelemesini kaldırırsak, `Main()` metodu onlara erişemez. Genel olarak, sınıflarda olduğu gibi, yapılardaki öğelere, yapı içinden engelsiz erişilir. Ama yapı dışından erişilebilmesi için yeterli bir erişim belirtecine sahip olması gerekir. Dışarıdan erişilmesini istemediğimiz değişkenleri bu izni vermeyerek onları gizlemiş (kapsüllemiş) oluruz.

İç-içe Yapılar

İç-içe yapılar kurulabilir. Aşağıdaki programda en dıştaki `Yapılar` adlı yapının içinde `Kimlik` ve `Uygulama` adlı yapılar yuvalanmıştır. `Main()` metodu `Uygulama` adlı yapının içindedir.

Yapı07.cs

```
using System;
struct Yapılar
{
    struct Kimlik
    {
        public string ad;
        public int yaş;
    }

    struct Uygulama
    {
        static void Main(string[] args)
        {
            Kimlik k;
            k.ad = "Pınar";
            k.yaş = 20;
            Console.WriteLine(k.ad);
            Console.WriteLine(k.yaş);
        }
    }
}
```

Yapılar İçinde Metotlar

Önceki programlarda yapı içinde `Main()` metodunun çalıştığını gördük. Aşağıdaki programda yapı içinde iki alan ve üç metot tanımlıdır. Yapıya ait nesnede alanlara değer atanmakta ve sonra atanan değerlerle işlem yapılmaktadır.

Yapı08.cs

```
using System;
struct Yapılar
{
    int x ;
    int y ;

    public void Ver(int i, int j)
    {
        x = i;
        y = j;
    }
```

```
    public void ToplamYaz()
    {
        int toplam = x + y;
        Console.WriteLine("Toplam = {0}", toplam);
    }

    public static void Main()
    {
        Yapılar aaa = new Yapılar();
        aaa.Ver(15, 25);
        aaa.ToplamYaz();
    }
}
```

Yapının static Değişkenleri

Sınıflarda dinamik değişkenlerin sınıfa ait her nesne içinde ayrı ayrı yaratıldığını biliyoruz. Bu demektir ki, sınıfa ait dinamik bir değişkenin herbir nesne içinde ayrı bir adresi vardır. Bir nesne içinde değişkene atanan değer öteki nesnelerin içindeki değerleri etkilemez. Ancak, statik değişkenlerde durum farklıdır. Sınıfa ait statik bir değişken nesneler içinde yaratılamaz; onun ana bellekte bir tek adresi olur. Bu adres sınıfa ait nesnelerden bağımsızdır. Dolayısıyla, herhangi bir anda statik değişkenin bir tek değeri vardır, o da en son atanan değerdir.

Bu olgu yapılar için de aynen geçerlidir. `static` nitelemeli değişkenler yapıya ait nesneler içinde yaratılmaz. Yapıya ait kaç nesne yaratılırsa yaratılsın, statik değişkenin bir tek adresi ve dolayısıyla herhangi bir anda bir tek değeri vardır, o da en son atanan değerdir. Aşağıdaki program bu gerçeği göstermektedir.

Yapı09.cs

```
using System;
struct Yapılar
{
    static int x = 250;
    static int y = 500;

    public void Ver(int i, int j)
    {
        x = i;
        y = j;
    }

    public void Ver(int i)
    {
        x = i;
        y = i;
    }

    public static void Main()
    {
        Console.WriteLine("x = {0} , y = {1} " ,x,y);
            Yapılar aaa = new Yapılar();
            aaa.Ver(10, 20);
            Console.WriteLine("x = {0} , y = {1} " ,x,y);
        Yapılar bbb = new Yapılar();
```

```
            bbb.Ver(1000);
            Console.WriteLine("x = {0} , y = {1} " ,x,y);


        }
}
```

Çıktı

```
x = 250 , y = 500
x = 10 , y = 20
x = 1000 , y = 1000
```

Programı çözümleyelim.

Yapının `x` ve `y` adlı iki `alanı (field)` var. Bu alanlar `static` nitelemeli olduğu için, bildirim anında, sırasıyla 250 ve 500 ilk değerleri atanabilmiştir. `Main()` metodunun ilk deyimi bu değerleri konsola yazdırıyor: x = 250 , y = 500.

Sonra `Main()` içindeki

```
Yapılar aaa = new Yapılar();
aaa.Ver(10, 20);
Console.WriteLine("x = {0} , y = {1} " ,x,y);
```

deyimleri çalışınca `aaa` adlı bir nesne yaratılıyor. Bu nesne içindeki `Ver` metodu `(10,20)` parametreleriyle çağrılıyor. Bu metot static değişkenlere x=10 ve y=20 atamalarını yapıyor. Bu yeni atamalar, statik değişkenlerin ilk değerlerinin yerine geçiyor. Böylece, son satır konsola bu yeni değerleri yazıyor: x = 10 , y = 20.

En son olarak `Main()` metodunun

```
Yapılar bbb = new Yapılar();
bbb.Ver(1000);
Console.WriteLine("x = {0} , y = {1} " ,x,y);
```

deyimleri çalışınca `bbb` adlı başka bir nesne yaratılıyor. Bu nesne içindeki `Ver` metodu `(1000)` parametresiyle çağrılıyor. Bu metot static değişkenlere x=1000 ve y=1000 atamalarını yapıyor. Bu yeni atamalar, statik değişkenlerin önceki değerlerinin yerine geçiyor. Böylece, son satır konsola bu yeni değerleri yazıyor: x = 1000 , y = 1000.

Bu programda dikkatimizi çeken başka bir şey daha var:

```
void Ver(int i, int j)
void Ver(int i)
```

metotları aynı ad ve aynı değer kümesine (void) sahiptirler. Böyle metotlara aşkın (overloaded) metotlar diyorduk. Bu program bize gösteriyor ki, yapılar içinde de aşkın metotlar tanımlanabilir.

Yapılar ve Kurucular

Sınıfların olduğu gibi yapıların da kurucuları tanımlanabilir. Parametresiz kurucular *genkurucu*'durlar (default constructor), onları ayrıca tanımlamaya gerek yoktur. Parametreli kurucular *aşkın* (overloaded) olabilirler. Aşağıdaki program bu gerçeği sergiler.

Yapı10.cs

```
using System;
struct Yapılar
{
```

```
    int x;
    int y;

    public Yapılar(int i, int j)
    { x = i; y = j; }

    public Yapılar(int i)
    { x = y = i; }

    public void AlanYaz()
    { Console.WriteLine("x = {0} , y = {1}", x, y); }
}
struct Uygulama
{
    public static void Main()
    {
        Yapılar aaa = new Yapılar(100, 200);
        Yapılar bbb = new Yapılar(300);
        aaa.AlanYaz();
        bbb.AlanYaz();
    }
}
```

Çıktı

```
x = 100 , y = 200
x = 300 , y = 300
```

Yapının içinde iki tane dinamik değişken ile iki tane aşkın kurucu tanımlanmıştır.

`Main()` metodu iki parametreli olan `Yapılar(100, 200)` kurucusu ile `aaa` adlı nesneyi kuruyor ve parametre değerlerini, sırasıyla `x` ve `y` dinamik değişkenlerine atıyor. `aaa.AlanYaz()` metodu konsola `x = 100 , y = 200` yazıyor.

Sonra `Main()` metodu tek parametreli olan `Yapılar(300)` kurucusu ile `bbb` adlı nesneyi kuruyor ve parametre değerini hem `x` hem `y` dinamik değişkenlerine atıyor. `bbb.AlanYaz()` metodu konsola `x = 100 , y = 200` yazıyor.

Struct ve Kalıtım

`struct` yapısı `System.ValueType` 'dan, `sınıflar` ise `System.Object`'den türerler. `struct` başka bir sınıf veya struct'tan türeyemez, türetilemez. Bunun yerine struct'un oluşturduğu (implemet) bir kaç önemli arayüzü vardır.

`struct`'u arayüz olarak kullanmak istersek, istemsiz (implicit) olarak kutulanır (boxed), çünkü arayüzler yalnızca referans tipleriyle iş yapabilirler. Örneğin,

Yapı11.cs

```
using System;
interface Iarayüz
{
    void birMetot();
}
struct Karma : Iarayüz
{
```

```
    public void birMetot()
    {
        Console.WriteLine("Struct Metodu çağrıldı");
    }
}
class Uygulama
{
    public static void Main()
    {
        Karma krm = new Karma();
        krm.birMetot();
    }
}
```

programında

```
Karma krm = new Karma();
```

deyimi Karma'nın bir nesnesini yaratır ve kutular (boxed). Arayüzün bütün metotları kutulanan bu nesne üzerinde işlevlerini yaparlar.

`class` ve `struct` yapısını bildiğimize göre, ne zaman `struct` kullanmak avantajlıdır, ne zaman değildir? sorusuna yanıt verebiliriz.

Ne zaman struct kullanmalı?

- Yaratılacak veri tipinin temel veri tipleri gibi davranması isteniyorsa.
- Çok sayıda nesne yaratılıp, hemen işi bitenler bellekten silinecekse. Örneğin, bir döngü içinde böyle bir durumla karşılaşılabilir.
- Yaratılan nesneler çok kullanılmıyor ve bellekte kalmaları gerekmiyorsa.
- Onu başka tiplerden türetmiyor ve ondan başka tipler türetilmiyorsa.
- Tutulan verinin başkalarına değer olarak aktarılabilmesi (pass by value) isteniyorsa.

Ne zaman struct kullanılmamalı?

- Struct'un öğelerinin bellekte işgal edecekleri bölge büyükse. (Microsoft bu büyüklüğü 16 byte ile sınırlandırmayı tavsiye ediyor).
- Yaratılan nesneler bir loleksiyona dahil ediliyor ve orada değiştiriliyor ya da onlarla döngü yapılıyorsa. Her işlemde boxing/unboxing yapılacağı için performans düşecektir. (Boxing/unboxing konusu ileride anlatılacaktır.

Sonuç

C# programcıya kendi veri tipini yaratmak için `struct` yapısı ile yeni bir araç sunmuştur. Her aracın iyi kullanılabileceği ve kullanılamayacağı yerler vardır. `Sınıf`'ın ve `struct`'un niteliklerini bilen programcı, yaratacağı veri tipini ne zaman sınıf ile ne zaman struct ile temsil edeceğine doğru karar vermelidir. Bazan birisi ötekinden daha etkin olabilir.

Bölüm 14

Özgenler (properties)

Özgen (property) nedir?

Get /set metotları

Yalnız-okunur (read-only) özgen

Yalnız-yazılır özgen

Erişimciler (accessors)

C# sınıf değişkenlerini *veri alanı (alan, `field`)* ve *özgen (özellik, `property`)* diye ikiye ayırır. `Alan (field)` ana bellekte kendisine bir yer ayrılan normal bir değişkendir. `Özgen (property)` ise alan ile metot arasında bir yerdedir. Alan özeliklerini taşır, onunla aynı rolü oynar; yani veri tutan bir bellek adresidir. Bu yönüyle alan sayılır. Öte yandan, metodun bazı özeliklerine sahiptir; tutacağı verinin giriş ve çıkışını bir metot gibi yapar. Bu yönüyle bakınca bir metottur. Bu nedenle, bazı kaynaklar, özgene akıllı alan (smart field) derler. Özgen' nin ne olduğunu anlamanın en iyi yolu onu kullanmaktır. Aşağıda o işi yapacağız.

Skorlar.cs

```
using System;
class Skorlar
{
    private int skor;
    public int Skor
    {
        get
        {
            return skor;
        }
        set
        {
            skor = value;
```

```
        }
    }
}
```

`Skorlar` sınıfının iki öğesi (class member) var. private olan birincisine dışarıdan erişilemez ve alışkın olduğumuz bir yöntemle bildirilmiş:

```
private int skor;
```

Ama `public` nitelemeli ikinci bildirim

```
    public int Skor
    {
        get
        {
            return skor;
        }
        set
        {
            skor = value;
        }
    }
```

kodlarıyla bildirilmiş. Ancak bu kodlar şimdiye kadar öğrendiklerimizle hemen çözümlenemiyor. Bu bildirim metot bildirimine benziyor, ama metot değil; çünkü `Skor` 'un sonunda `()` metot parantezi yok. Kurucu olmadığı da açık. Skor blokunun içinde `get` ve `set` blokları sanki birer metot gibi, ama başlıkları (imzaları) alışılmış metot tanımına uymuyor. Get bloku, '`return skor;`' deyimi ile sınıfın private nitelemeli değişkenini okuyor. Set bloku ise '`skor = value;`' deyimi ile sınıfın private nitelemeli değişkenine '`value`' değerini atıyor, ancak '`value`' sınıf içinde tanımlı değil.

Çok karmaşık görünen bu bildirim, adına `özgen (property)` denilen özel bir tür *(akıllı alan – smart field)* bildiriminden başka bir şey değildir. *private* olan `skor` değişkenine değer atama ve okuma eylemlerini public olan `Skor` değişkeni yapıyor. Böylece sınıf dışından erişilemeyen *private* nitelemeli değişkene dışarıdan erişme olanağı yaratılıyor. `set` blokundaki '`value`' C# dilinde bir anahtar sözcüktür. İşlevi burada olduğu gibi dışarıdan erişilemeyen değişkene değer atamaktır. Atayacağı değer, `Skorlar` sınıfına ait bir *nesne* yaratılınca public nitelemeli `Skor` değişkenine atanacak değerdir.

Bu bildirimdeki `get` ve `set` bloklarına getter/setter metotları ya da erişimciler (accessors) denilir. Türkçe'de *getir/götür* veya *ver/al* diyebileceğimiz özel iki metottur. Anahtar sözcükler gibi kullanıldığı için, biz `getter/setter` sözcüklerini kullanacağız.

Artık private değişkenimize değer vermeyi deneyebiliriz.

```
class Uygulama
{
    static public void Main()
    {
        Skorlar yeniSkor = new Skorlar(); // nesneyi yarat
        yeniSkor.Skor = 73; // Skor'a setter metodu ile değer ata
        int ySkor = yeniSkor.Skor; //getter metodu ile Skor'u oku

        Console.WriteLine(" Skor : {0}" , ySkor);


    }
}
```

Aşağıdaki program bir sınıfın iki nesnesini yaratıyor ve onların private değişkenlerine erişimcilerle değer atıyor.

Kişi.cs

```
using System;
class Kişi
{
    private int yaş;
    private string doğumYeri;
    public int Yaş
    {
        get
        {
            return yaş;
        }
        set
        {
            if (value <= 65 && value >= 18)
            {
                yaş = value;
            }
            else
                yaş = 18;
        }
    }
    public string DoğumYeri
    {
        get
        {
            return doğumYeri;
        }
        set
        {
            doğumYeri = value;
        }
    }
}

class Uygulama
{
    static void Main(string[] args)
    {
        Kişi Mehmet = new Kişi();
        Kişi Dilek  = new Kişi();

        Mehmet.Yaş = 21;
        Mehmet.DoğumYeri = "Malatya";
        Dilek.Yaş = 18;
        Dilek.DoğumYeri = "İzmir";

        Console.WriteLine("Mehmet'in yaşı = {0}, doğum yeri = {1}",
Mehmet.Yaş, Mehmet.DoğumYeri);
        Console.WriteLine("Dilek'in yaşı  = {0}, doğum yeri = {1}",
Dilek.Yaş , Dilek.DoğumYeri );
    }
}
```

Çıktı

```
Mehmet'in yaşı = 21, doğum yeri = Malatya
Dilek'in yaşı  = 18, doğum yeri = İzmir
```

Veri Kapsülleme ve Veri Saklama

Nesne Yönelimli Programlamada veri kapsülleme ve veri saklama çok önemli iki kavramdır. Veri kapsüllemeyi sınıf (class) içinde veya yapı (struct) içinde yaparız. Erişim belirteçleri denilen private, public, protected, internal gibi nitelemelerle, dışarıdan veriye erişimi izne bağlarız. Özgenlerin asıl işlevi, bir sınıfta başkaları tarafından görünmesini istemediğimiz verileri tutmaktır. Bu yönüyle, özgen, güvenliği sağlanmış bir veri alanıdır. (Tabii, her veri alanının bir değişken olduğunu unutmuyoruz.) Sözkonusu güvenlik çok basit bir yöntemle sağlanır. C# bir veri alanındaki veriyi korumaya almak istiyorsa, o veri alanına veri girişini ve girilen veriyi okumayı özgen (property) ile yapar. Yukarıdaki örnekler Veri Kapsülleme (field encapsulation) ve veri saklamaya örnektirler. Bu konu, ileride yeri geldikçe tekrar ele alınacaktır.

Bölüm 15

String Sınıfı

String Sınıfı

`String`, `.NET Framework` içinde `System` aduzayına (namespace) ait bir sınıftır. Dolayısıyla `System.String` sınıfını C# olduğu gibi alır ve kullanır. Klâsik dillerde *string* karakterlerden oluşan bir *array* olarak tanımlanır. `System.String` sınıfı da o geleneği sürdürür, ama çok farklı nitelikler ekleyerek. Bir string (nesnesi) unicode karakterlerden oluşan metindir (text). Başka bir deyişle `System.Char` nesnelerinin bir dizisidir. Ama bu dizi bir bütündür, parçalanamaz, değiştirilemez. Bir stringe *ekleme, birleştirme, ayırma, kırpma (trim), silme, yerine koyma (replace)* gibi klâsik metin işlemleri uygulanamaz. Bu işlemlerden herhangi birisi uygulandığında başka bir string yaratılmış olur. Gerçekten, aşağıda göreceğimiz ve string üzerinde değişiklik yapan metotlar, söz konusu stringi (değerini) değiştirmezler, onun yerine her değişiklik için yeni bir string yaratırlar.

`String` sınıfı, girdi/çıktı eylemlerinde yaşamsal role sahiptirler. Metinler, sayılar, tarihler gibi farklı veri tipleri bilgisayara string olarak girer, bilgisayardan string olarak çıkarlar. Kullanıcı ile bilgisayarın etkileşim içinde olabilmesi için, giriş işleminde *string 'den veri tipine*, çıkışta ise *veri tipinden string'e* gerekli dönüşümler yapılır. `String` 'in bu önemli işlevleri nedeniyle, `.NET Framework` string işlemleri yapmaya yarayan çok sayıda metoda sahiptir. Bunlara ek olarak çok sayıda arayüz ile `StringBuilder`, `String.Format`, `StringCollection`, vb. sınıflar da string işlemlerinin yapılmasını kolaylaştırırlar. Bunların hepsini sistematik olarak açıklamak bu kitabın amacı dışındadır. Konuyu aydınlatmaya yetecek örnekler vermekle yetineceğiz. Zaten önceki bölümlerde `string` kavramını çok kullandık. Sonraki bölümlerde de farklı niteliklerini kullanmaya devam edeceğiz. Bütün bunların ötesinde `string` 'in sistematik yapısını incelemek isteyenler msdn web sitesine bakabilirler.

.NET Framework, `System.String` sınıfına kısa bir takma ad veriris `string`. Biz de bu ikisini eşanlamlı kullanacağız; yani `System.String` ve `string` aynı anlama sahiptirler.

Öte yandan, string çok özel bir sınıftır. Diğer sınıflarda olmayan nitelikleri vardır. Çok sayıdastatik öğesi vardır; dolayısıyla, ona ait bir nesne yaratmaya gerek kalmaksızın onları kullanabiliriz. Örneğin,

```
string str = "Ankara";
```

deyimini yazdığımızda `str` referansı tarafından işaret edilen bir string nesnesi yaratıp ona değer atamış oluruz; ayrıca `new` operatörünü kullanmaya gerek yoktur. Böyle olduğu için `string` bir referans tipidir, *string tipi* bir nesneyi işaret eder. Bir nesne işaret etmediği zaman değeri `null` olur; yani hiç bir adresi göstermez. Bu demektir ki,

```
string str = null ;
```

deyimi geçerlidir.

String Sınıfının Özgenleri (properties)

String.Chars : Bir string içinde belirlenen bir yerdeki karakteri söyler.

String.Length : string'in uzunluğunu; yani kaç karakterden oluştuğunu söyler.

Aşağıdaki program, bir stringin uzunluğunu ve buluyor ve stringi bir karakter arrayi gibi alıp öğelerini sırayla yazıyor

StrLength.cs

```
using System;

class StrLength
{
    public static void Main()
    {
        string s = "Bilgimiz gücümüzdür!";
        for (int i = 0; i < s.Length; i++)
        {
            Console.Write("s[" + i + "]  = " + s[i]);
        }
    }
}
```

String Sınıfının Metotları

String sınıfının 50 den çok metodu vardır. Bu metotlar, stringlerle yapılabilecek *mukayese, ekleme, insert, dönüştürme, copyalama, formatlama, indexleme, birleştirme, ayırma, doldurma, kırpma, silme, yerine koyma ve arama* gib eylemleri yapmamızı sağlarlar. Aşağıda bunların bazı örneklerini göreceğiz.

String Bildirimi

`string` sınıfı, .NET Framework içinde öntanımlı bir sınıftır. Anımsayacağınız gibi, öntanımlı sınıflar klâsik dillerdeki veri tipi gibi kullanılabilirler; yani onlara ait nesneleri bizim yaratmamıza gerek yoktur; kullanacağımız metotlarının çoğu statiktir. Dolayısıyla, *string bildirimi* için şu yolları izleyebilirizis

```
string birMetin = "Ankara başkenttir"; // metin
String birsayı = "123456789"; // sayı
System.String birTarih = "15.08.2008"; // tarih
string kod = "\\\u0084\n";  // backslash,  ? işareti, satırbaşı
```

Metot Örnekleri

Yukarıdaki bildirimlerin tiplerinin `System.String` olduğunu görmek için `GetType()` metodunu kullanabilirizis

String01.cs

```
using System;

namespace Stringler
{
    class String01
    {
        static void Main(string[] args)
        {
            string birMetin = "Ankara başkenttir";
            String birSayı = "123456789";
            System.String birTarih = "15.08.2008";

            Console.WriteLine(birMetin.GetType());
            Console.WriteLine(birSayı.GetType());
            Console.WriteLine(birTarih.GetType());
        }
    }
}
```

Çıktı

```
System.String
System.String
System.String
```

String nesnedir, değiştirilemez (immutable)

Bildirimi yapılan bir string üzerinde herhangi bir değişiklik yapılamayacağını, yapılan her değişikliğin başka bir string yarattığını söylemiştik. Bunu aşağıdaki örnekten görebiliriz. `str1` stringi `StringDeğiştir()` metodu ile değiştirilmek isteniyor, ama değişiklik olmuyor, ona dokunulmadan başka bir string yaratılıyor; orijinal `str1` değişmeden kalıyor.

Değişmez.cs

```
using System;

class Değişmez
```

```
{
    static void Main(string[] args)
    {
        string str1 = "Doğayı koru!" ;
        Console.WriteLine("str1 in ilk değeriis {0}", str1);
        Console.WriteLine(StringDeğiştir(str1));
        Console.WriteLine("str1 in son değeriis {0}", str1);
    }

    static string StringDeğiştir(string str)
    {
        str = "Doğayı kimler yokediyor?";
        return str;

    }
}
```

Çıktı

```
str1 in ilk değeriis Doğayı koru!
Doğayı kimler yokediyor?
str1 in son değeriis Doğayı koru!
```

Stringleri Birleştirme

Stringleri birleştirmek için değişik yöntemler kullanabiliriz. Sayılar için tanımlanan (+) ve (+=) operatörleri string tipi için *override* edilmişlerdir. Bunlar kullanılabileceği gibi, `String` sınıfının bir öğesi olan `Concat()` metodu da kullanılabilir.

Aşağıdaki üç deyim aynı işi yaparis

```
string s1 = "Bu " + "gün " + "hava " + "çok sıcak" + ".";
```

```
string s2 = "Bu ";
       s2 += "gün ";
       s2 += "hava ";
       s2 += "çok sıcak ";
       s2 += ".";
```

```
string s3 = String.Concat("Bu ", "gün ", "hava ", "çok sıcak", ".");
```

Gerçekten aynı olduklarını görmek istersek, bunları bir program içine alıp sonuçları yazdırabilirizis

Birleştir01.cs

```
using System;

class Birleştir01
{
    public static void Main()
    {
        string s1 = "Bu " + "gün " + "hava " + "çok sıcak" + ".";
```

```
        string s2 = "Bu ";
        s2 += "gün ";
        s2 += "hava ";
        s2 += "çok sıcak ";
        s2 += ".";

        string s3 = String.Concat("Bu ", "gün ", "hava ", "çok sıcak", ".");
        Console.WriteLine(s1);
        Console.WriteLine(s2);
        Console.WriteLine(s3);

    }
}
```

Çıktı

```
Bu gün hava çok sıcak.
Bu gün hava çok sıcak .
Bu gün hava çok sıcak.
```

Benzer işi, değişkenleri kullanarak da yapabiliriz. `GetType()` metodu veri tipini belirtir.

Birleştir02.cs

```
using System;
namespace Stringler
{
    class Birleştir02
    {
        static void Main(string[] args)
        {
            string s1 = "Ankara";
            Console.WriteLine(s1);
            System.String s2 = "başkenttir";
            String boşluk = "  ";

            Console.WriteLine("s1 = " + s1);
            Console.WriteLine("s2 = " + s2);
            Console.WriteLine("s1 + boşluk + s2 =  " + s1 + boşluk + s2);
            Console.WriteLine("s1.GetType()      = "  + s1.GetType());
            Console.WriteLine("s2.GetType()      = "  + s2.GetType());
            Console.WriteLine("boşluk.GetType() = " + boşluk.GetType());
        }
    }
}
```

Çıktı

```
Ankara
s1 = Ankara
s2 = başkenttir
s1 + boşluk + s2 =  Ankara  başkenttir
s1.GetType()      = System.String
s2.GetType()      = System.String
boşluk.GetType() = System.String
```

Stringleri Karşılaştırma

Genel olarak, iki referans tipinin (reference type) karşılaştırılması demek, işaret ettikleri *bellek adreslerinin* aynı olup olmadığının belirlenmesi demektir. Bellek adreslerinde yazılı veriye bakılmaz. Tersine olarak, iki değer tipinin (value type) karşılaştırılması demek, *bellek adreslerindeki verilerin* aynı aynı olup olmadığının belirlenmesi demektir. Zaten değer tipinden farklı iki değişkenin adresleri farklı olacağı için, onların adreslerini karşılaştırmanın bir anlamı olamaz; onun yerine o farklı adreslerde yazılı verilerin eşit olup olmadıklarına bakılır.

`String` de referans tipi olduğu için, iki stringin karşılaştırılmasında, işaret ettikleri adreslerin aynı olup olmadığı belirlenir kanısına varmak gerekiyor. Oysa, iki stringin karşılaştırılmasında bellek adreslerinin aynı olup olmadığına değil, onları oluşturan karakter dizilerinin aynı olup olmadığına bakılır, yani değer tipinde olduğu gibi bellek adreslerinde yazılı olan değerler karşılaştırılır. Stringlerin karşılaştırılmasında ortaya çıkan bu özelik, onların değer tipi imiş gibi algılanmasına neden olabilir. Ancak, bu özeliğin bilinçli olarak yapıldığını söylemeliyiz. Stringleri karşılaştırırken, kullanıcı onların bellek adresleriyle değil, değerleriyle ilgilidir.

Stringleri karşılaştırmak için `==` ile `!=` ilişkisel operatörlerini ya da `Compare()` metodunu kullanabiliriz. İlşkisel operatörler yalnızca stringlerin eşit olup olmadığını `True` veya `False` değerleriyle belirtir. `Compare()` metodu ise çok daha fazla işler yapar. Aşağıdaki iki örnek bunları göstermektedir.

Mukayese01.cs

```
using System;
namespace Stringler
{
    class Mukayese01
    {
        static void Main(string[] args)
        {
            string s1 = "Ankara";
            string s2 = "istanbul";
            string s3 = "ANKARA";
            string s4 = "Ankara";

            Console.WriteLine(s1 == s2); // False
            Console.WriteLine(s1 != s2); // True
            Console.WriteLine(s1 == s4); // True
            Console.WriteLine(s1 == s3); // False
            Console.WriteLine("Ankara" == "ANKARA"); // False
            Console.WriteLine("Ankara" != "ANKARA"); // True

        }
    }
}
```

Aşağıdaki programda `s1` ve `s2` stringlerinin içerikleri aynı, adresleri farklıdır. `s1 == s2` karşılaştırmasının `True`, ama object olarak `(object)s1 == (object)s2` karşılaştırmasının `False` sonuç verdiğine dikkat ediniz.

Mukayese02.cs

```
using System;
namespace Stringler
```

```
{
    class Mukayese02
    {
        static void Main(string[] args)
        {
            string s1 = "Ankara";
            string s2 = "An";
            s2 += "kara"; // s1, s2 farklı adreste
            Console.WriteLine(s1.ToString());
            Console.WriteLine(s2);
            Console.WriteLine(s1 == s2);
            Console.WriteLine((object)s1 == (object)s2);
            Console.WriteLine((object)s1) ;
            Console.WriteLine((object)s2);
        }
    }
}
```

Çıktı

```
Ankara
Ankara
True
False
Ankara
Ankara
```

Compare() Metodu

Stringleri karşılaştırmak için `string` sınıfının bir öğesi olan `Compare()` metodunu kullanarak `==` ile `!=` ilişkisel bağıntılarının verdiğinden fazlasını elde edebiliriz. Gerçekten bu metot, farklı diller, farklı alfabeler ve farklı kültürlerde yazılabilecek stringleri karşılaştırır; eşit olup olmadıklarını; eşit değillerse hangisinin büyük, hangisinin küçük olduğunu bildirir. Dünyadaki bütün diller, bütün alfabeler ve bütün kültürler için işlevselliği olan bu metot default olarak Windows İşletim Sisteminin diline ve kültürüne bağlıdır. Türkçe Windows kullanıyorsak, Türk alfabesine göre karşılaştırma yapacaktır.

Yapılan karşılaştırma, ilgili kültürün alfabetik sıralamasına (sözlük sıralaması) göredir.

`Compare()` metodunun değer kümesi `Int32` veri tipidir. Bu demektir ki, `s1` ile `s2` stringlerini mukayese etmek için

```
String.Compare(s1,s2) ;
```

deyimini yazarsak, metodun alacağı değer işaretli bir `int` tipi sayıdır. Eğer bu sayı negatif ise `s1 < s2` dir, sıfır ise `s1 = s2` dir, pozitif ise `s1> s2` dir. Bu söylediklerimizi aşağıdaki tabloda gösterebiliriz.

Compare(s1,s2)	Anlamı
Negatif	`s1 < s2`

Sıfır	`s1 = s2`
Pozitif	`s1 > s2`

Karşılaştırılan stringlerden birisi veya her ikisi de `null` ya da `boş` (`""`) string olabilir. `null` bütün stringlerden küçüktür. *Boş string* ise, boş olmayan her stringden küçüktür. İki `null` ya da iki *boş string* karşılaştırılırsa 0 (eşit) değeri çıkar.

Mukayese03.cs

```
using System;
namespace Stringler
{
    class Mukayese03
    {
        static void Main(string[] args)
        {
            string s1 = "Ankara";
            string s2 = "istanbul";
            string s3 = "ANKARA";
            string s4 = "Ankara";

            Console.WriteLine(String.Compare(s1,s2));       // -1
            Console.WriteLine(String.Compare(s1, s3));      // -1
            Console.WriteLine(String.Compare(s1, s4));      //  0
            Console.WriteLine(String.Compare(null, s2));    // -1
            Console.WriteLine(String.Compare(s1, null));    //  1
            Console.WriteLine(String.Compare(null, null));  //  0
            Console.WriteLine(String.Compare("", s2));      // -1
            Console.WriteLine(String.Compare(s1, ""));      //  1
            Console.WriteLine(String.Compare("", ""));      //  0

        }
    }
}
```

Equals() Metodu

İki stringin eşit olup olmadığını anlamak için `Equals()` metodunu da kullanabiliriz. Aşağıdaki programda bu metodun *overload* edilmiş iki sürümünü göreceksiniz. Programı satır satır çözümleyiniz.

Equal01.cs

```
using System;

class Equal01
{
    public static void Main()
    {
        bool b;
        string s1 = "abcd";
        string s2 = "abcde";
```

```
        b = String.Equals("abc", "abc");
        Console.WriteLine("String.Equals(\"abc\", \"abc\") = " + b);

        b = s1.Equals(s2);
        Console.WriteLine("s1.Equals(s2) = " + b);

        b = s1 == s2;
        Console.WriteLine("s1 == s2 = " + b);
    }
}
```

Çıktı

```
String.Equals("abc", "abc") = True
s1.Equals(s2) = False
s1 == s2 = False
```

Copy() metodu

Bir stringi başka bir stringe kopyalamak için `Copy()` metodunu kullanırız. Aşağıdaki program `s2` stringini `s1` stringine kopyalıyor.

Kopya01.cs

```
using System;

class Kopya01
{

    public static void Main()
    {
        string s1 = "Ankara";
        string s2 = "C# öğreniyorum";
        Console.WriteLine("s1 = " + s1);
        Console.WriteLine("s2 = " + s2);
        Console.WriteLine("s2 stringi s1 'e kopyalanıyor");
        s1 = String.Copy(s2);
        Console.WriteLine("s1 = " + s1);
    }
}
```

Çıktı

```
s1 = Ankara
s2 = C# öğreniyorum
s2 stringi s1 'e kopyalanıyor
s1 = C# öğreniyorum
```

Insert(), Remove(), Replace() metotları

`Insert()` metodu bir string'in belirlenen bir yerine başka bir string'i yerleştirir. İki parametresi vardır. Birinci parametrede string'in nereye yerleştirileceğini belirten bir tamsayıdır. İkinci parametre ise birinci stringe yerleştirilecek olan string'dir.

Yerleştir01.cs

```
using System;

class Yerleştir
{
    public static void Main()
    {
        string str1 =  "C# kolaydır.";
        string str2 = " dilini öğrenmek çok ";
        Console.WriteLine("str1 = "  + str1);
        Console.WriteLine("str2 = " + str2);
        str1 = str1.Insert(3, str2);
        Console.WriteLine("str1 =  " + str1);
    }
}
```

Çıktı

```
str1 =  C# kolaydır.
str2 =  dilini öğrenmek çok
str1 =  C#  dilini öğrenmek çok kolaydır.
```

Replace() metodu

Bir string içindeki bir karakter veya bir alt-stringi bulur ve onun yerine istenen başka bir harf veya stringi koyar. İki parametresi vardır. Birincisi aranacak harf veya alt-string, ikincisi bulunanın yerine konulacak harf veya string.

Aşağıdaki program l harfi yerine h harfini koyar; sonuçta *Ceylan* stringi *Ceyhan* stringine dönüşür.

Replace01.cs

```
using System;

class Replace01
{
    public static void Main()
    {
        string str = "Ceylan";
        Console.WriteLine(str);
        str = str.Replace('l', 'h'); // l yerine h koy
        Console.WriteLine(str);
    }
}
```

Aşağıdaki program *"C# dili çok uzun zamanda öğrenilir."* stringinde *"uzun"* alt-stringi yerine *"kısa"* stringini koyar.

Replace02.cs

```
using System;

class Replace02
```

```
{
    public static void Main()
    {
        string str1 = "C# dili çok uzun zamanda öğrenilir.";
        string str2 = " kısa ";
        Console.WriteLine("str1 = " + str1);
        Console.WriteLine("str2 = " + str2);
        str1 = str1.Replace("uzun", str2);
        Console.WriteLine("str1 =  " + str1);
    }
}
```

Remove() Metodu

Bir stringden bir karakter veya bir alt stringi siler. İki tamsayı parametresi vardır. Birinci parametre silme eyleminin kaçıncı karakterden başlayacağını, ikinci parametre kaç karakter silineceğini gösterir.

Aşağıdaki program *"C# dili çok ama çok kısa zamanda öğrenilir."* stringinden *"ama çok"* alt-stringini siler ve *"C# dili çok kısa zamanda öğrenilir."* stringine dönüştürür.

Yoket01.cs

```
using System;

class Yoket01
{
    public static void Main()
    {
        string str1 = "C# dili çok ama çok kısa zamanda öğrenilir.";
        Console.WriteLine("str1 = " + str1);
        str1 = str1.Remove(12, 8);
        Console.WriteLine("str1 = " + str1);
    }
}
```

Çıktı

```
str1 = C# dili çok ama çok kısa zamanda öğrenilir.
str1 = C# dili çok kısa zamanda öğrenilir.
```

ToUpper() ve ToLower() Metotları

ToUpper() metodu bir stringdeki bütün harfleri büyük harf karşılıklarına dönüştürür. ToLower() metodu ise tersini yapar, bir stringdeki bütün büyük harfleri küçük harf karşılıklarına dönüştürür.

Aşağıdaki program bu iki dönüşümün kullanılışını göstermektedir.

UpperLower.cs

```
using System;

class UpperLower
{
```

```
    public static void Main()
    {
        string str1 =  "C# dili çok kısa zamanda öğrenilir.";
        Console.WriteLine("str1 = " + str1);
        // büyük harfe dönüştür
        string str2 = str1.ToUpper();
        Console.WriteLine("str1.ToUpper() =  " + str2);

        // küçük harfe dönüştür
        str2 = str2.ToLower();
        Console.WriteLine("str2.ToLower() =  " + str2);
    }
}
```

Çıktı

```
str1 = C# dili çok kısa zamanda öğrenilir.
str1.ToUpper() =  C# DİLİ ÇOK KISA ZAMANDA ÖĞRENİLİR.
str2.ToLower() =  c# dili çok kısa zamanda öğrenilir.
```

Split() ve Join() Metotları

Başka bir sistem veya programın girdi/çıktısı ile bağlantılı bir iş yaparken, farklı biçemdeki (format) verileri kullanmamız gerekebilir. Örneğin, bir çok programın çıktısı Excel biçemine dönüşebilir ve bir çok program Excel biçemindeki verileri okuyabilir. Excel *Comma Separated Value (CSV) format* denilen virgüllerle ayrılmış verileri girdi/çıktı olarak kabul eder. C# ile Excel arasında veri alışverişi yapabilmemiz için, örneğin, CVS biçemindeki verileri bir array'e dönüştürmek ya da tersine olarak bir array'i CVS biçemine dönüştürmek gerekecektir. Bu iş için Split ve Join() metotlarını kullanırız. Aşağıdaki örnekler, bu metotların nasıl kullanıldığını göstermektedir.

Aşağıdaki program bir string array'ini CVS biçemine dönüştürmektedir. [Array'in öğelerini (veriler) birbirinden ayırmak için virgül veya başka bir karakter kullanılabilir. Bu örnekte `'*'` kullanılmıştır.]

Join01.cs

```
  using System;

class MainClass
{
    public static void Main()
    {
        string[] dizi1 = { "C#", "ile", "Nesne", "Yönelimli",
"programlama", "öğreniyorum" };
        string dizi2 = String.Join("*", dizi1);
        Console.WriteLine("dizi2 = " + dizi2);
    }
}
```

Çıktı

```
dizi2 = C#*ile*Nesne*Yönelimli*programlama*öğreniyorum
```

Aşağıdaki program bir arrayi önce CVS biçemine dönüştürüyor, sonra onun virgülle birbirlerinden ayrılmış öğelerini (veriler) listeliyor.

Split01.cs

```
using System;

class MainClass
{
    public static void Main()
    {
        string[] dizi1 = { "C#", "ile", "Nesne", "Yönelimli",
"programlama", "öğreniyorum" };
        string dizi2 = String.Join(",", dizi1);
        Console.WriteLine("dizi2 = " + dizi2);

        dizi1 = dizi2.Split(',');
        foreach (string s in dizi1)
        {
            Console.WriteLine("s = " + s);
        }
    }
}
```

Çıktı

```
dizi2 = C#,ile,Nesne,Yönelimli,programlama,öğreniyorum
s = C#
s = ile
s = Nesne
s = Yönelimli
s = programlama
s = öğreniyorum
```

Aşağıdaki program CVS olarak verilen ayları öğelerine ayırıyor, sonra onları (;) ile birleştirip yazıyor. Ayrıca istenen bir mevsime ait ayları buluyor. Programı satır satır çözümleyiniz.

StrJoinSplit.cs

```
using System;

namespace SplitJoin
{
    class StrJoinSplit
    {
        static void Main(string[] args)
        {
            // CVS biçemindeki string
            string cvs =
"Ocak,Şubat,Mart,Nisan,Mayıs,Haziran,Temmuz,Ağustos,Eylül,Ekim,Kasım,Aral
ık";

            Console.WriteLine("Original CVS Stringiis \n{0}\n", cvs);

            // separate individual items between commas
            string[] sene = cvs.Split(new char[] { ',' });

            Console.WriteLine("Ayrılmış öğeleris ");

            foreach (string ay in sene)
```

```
            {
                Console.Write("{0} ", ay);
            }
            Console.WriteLine("\n");

            // Arrayin öğelerini ; ile birleştir
            string cvs2 = String.Join(";", sene);

            Console.WriteLine("; ile birleşen arrayis \n{0}\n", cvs2);

            string[] mevsim = cvs.Split(new Char[] { ',' }, 3);

            Console.WriteLine("İlk üç öğeis ");

            foreach (string ay in mevsim)
            {
                Console.Write("{0} ", ay);
            }
            Console.WriteLine("\n");

            string güz = String.Join("/", sene, 6, 3);

            Console.WriteLine("Güz mevsimiis \n{0}\n", güz);
        }
    }
}
```

IndexOf() Metodu

Bir string içindeki bir harf veya alt-stringin kaçıncı karakterden başladığını belirtir. Metodun birisi zorunlu üç parametresi vardır. Zorunlu olan birincisi aranan harf veya stringdir. İkincisi kaçıncı harften başlayarak arama yapılacağını, üçüncü parametre ise kaç harf boynca arama yapılacağını gösterir. Metod aranan bulamazsa -1 değerini alır.

IndexOf01.cs

```
using System;

class IndexOf01
{
    public static void Main()
    {
        string str1 =  "C# dili çok kısa zamanda öğrenilir.";
        Console.WriteLine("str1 = " + str1);
        // "kısa" stringinin yerini bulur
        Console.WriteLine("str1.IndexOf(\"kısa\") =  " +
str1.IndexOf("kısa"));
    }
}
```

Çıktı

```
str1 = C# dili çok kısa zamanda öğrenilir.
str1.IndexOf("kısa") =  12
```

Aşağıdaki program *"C# dili çok kısa zamanda öğrenilir."* stringi içinde `'i'` harfini 10-uncu karakterden sonra aramaya başlıyor ve aramayı 5 karakter boyunca sürdürüyor. Bulamadığı için -1 değerini alıyor. Programda üçüncü parametre olarak 21 ile 25 arasında bir sayı yazarsanız metodun değeri 30 olur; yani aradığı `'i'` harfinin yeri stringin 30-uncu karakteridir. Üçüncü parametre 25 den büyük olursa, string uzunluğu dışına taşıldığını belirten şu hata mesajını verir.

Unhandled Exceptionis System.ArgumentOutOfRangeExceptionis Count must be and count must refer to a location within the string/array/collection.

IndexOf02.cs

```
using System;

class IndexOf02
{
    public static void Main()
    {
        string str1 =  "C# dili çok kısa zamanda öğrenilir.";
        Console.WriteLine("str1 = " + str1);
        // "kısa" stringinin yerini bulur
        Console.WriteLine("str1.IndexOf('i',10,5 ) =  " +
str1.IndexOf('i',10,5));
    }
}
```

IndexOfAny() Metodu

Bir metinde istenmeyen harflerin olup olmadığını bilmek isteyebiliriz. Bu durumda `IndexOfAny()` metodunu kullanırız. Bu metot bir string'in içerisinde bir karakter array'ine ait herhangi bir karakterin olup olmadığını denetler; varsa bulduğu karakterin string içinde kaçıncı karakterden sonra geldiğini belirtir. Parametreleri `IndexOf()` fonksiyonu ile aynıdır. Metot aradığını bulduğu ilk yeri belirten tamsayı değerini alır; bulamazsa -1 değerini alır.

Aşağıdaki örnek, aranan (#) karakterinin *"Çok kısa zamanda C# dili öğrenilir."* stringinin 18-inci karakter olduğunu bildirir.

```
using System;

class IndexOf02
{
    public static void Main()
    {
        string str1 =  "Çok kısa zamanda C# dili öğrenilir.";
        Console.WriteLine("str1 = " + str1);
        char[] dizi = {'$','#','%','*'};
        int n = str1.IndexOfAny(dizi, 10, 24);
        Console.WriteLine(n);
    }
}
```

LastIndexOf()

`public int LastIndexOf(char, int, int)` metodu bir string içinde bir karakterin en sondaki yerini bulur. Metodun üç parametresi vardır. Birinci parametre aranan, ikinci parametre aramaya nereden başlanacağını, üçüncü parametre aramanın kaç karakter süreceğini belirtir.

LastIndexOf01.cs

```
using System;

class LastIndexOf01
{
    public static void Main()
    {
        string str = "Bu bir denemedir.";

        Console.WriteLine(str.LastIndexOf(' ', 5, 6)); //6
        Console.WriteLine(str.LastIndexOf(' ', 6, 6)); //6
        Console.WriteLine(str.LastIndexOf(' ', 7, 6)); //6
        Console.WriteLine(str.LastIndexOf(' ', 8, 6)); //6
        Console.WriteLine(str.LastIndexOf(' ', 9, 6)); //6
        Console.WriteLine(str.LastIndexOf(' ',10, 6)); //6
        Console.WriteLine(str.LastIndexOf(' '));       //6
    }
}
```

PadLeft() ve PadRight() metotları

Bir stringin soluna ya da sağına istenildiği kadar başka karakter veya boşluk koyarlar.

Aşağıdaki örnekte stringin soluna ve saına birer kez boş karakter, birer kez de '*' karakteri konulmuştur.

Padding01.cs

```
using System;

class Padding01
{

    public static void Main()
    {
        string str = "C# ile Nesne Programlama";
        Console.WriteLine("str.PadLeft(30)       = |{0}|" ,
str.PadLeft(30));
        Console.WriteLine("str.PadLeft(30, '*')  = |{0}|" ,
str.PadLeft(30, '*'));
        Console.WriteLine("str.PadRight(30)      = |{0}|" ,
str.PadRight(30));
        Console.WriteLine("str.PadRight(30,'*')  = |{0}|" ,
str.PadRight(30,'*'));
    }
}
```

Çıktı

```
str.PadLeft(30)       = |       C# ile Nesne Programlama|
```

```
str.PadLeft(30, '*')   = |******C# ile Nesne Programlama|
str.PadRight(30)       = |C# ile Nesne Programlama      |
str.PadRight(30,'*')   = |C# ile Nesne Programlama******|
```

Trim(), TrimStart(), and TrimEnd() metotları

`Trim()` metodu parametresiz olarak kullanıldığında string'in önündeki ve gerisindeki boşluk karakterlerini (white space) siler. `TrimStart()` metodu string'in önündeki boşluk karakterlerini siler. `TrimEnd()` metodu string'in gerisindeki boşluk karakterlerini siler. Silinmesi istenen karakterler arrayi bir parametre olarak kullanılabilir.

Aşağıdaki program çeşitli kırpma yöntemlerini göstermektedir.

Trim01.cs

```
using System;

public class Trim01
{
    public static void Main()
    {
        string s1 = "    abc     ";
        Console.WriteLine("s1        = |{0}|" , s1);
        Console.WriteLine("s1.Trim() = |{0}|" , s1.Trim());

        string s2 = "__...,, ABC,..._";
        Console.WriteLine("s2 = " + s2);
        char[] trimEdilecek = new char[] { '.', ',', '_' };
        Console.WriteLine("trimEdilecek = { '.', ',', '_' }");
        Console.WriteLine("s2.Trim(trimEdilecek)      = |
{0}|",s2.Trim(trimEdilecek));
        Console.WriteLine("s2.TrimStart(trimEdilecek) = |{0}|"
,s2.TrimStart(trimEdilecek));
        Console.WriteLine("s2.TrimEnd(trimEdilecek)   = |{0}|",
s2.TrimEnd(trimEdilecek));
    }
}
```

Çıktı

```
s1           = |    abc     |
s1.Trim()    = |abc|
s2           = __...,, ABC,..._
trimEdilecek = { '.', ',', '_' }
s2.Trim(trimEdilecek)      = | ABC|
s2.TrimStart(trimEdilecek) = | ABC,..._|
s2.TrimEnd(trimEdilecek)   = |__...,, ABC|
```

Substring() metodu

`Substring()` metodu bir string'in içerisinde başlangıç yeri ve uzunluğu belitilen alt-stringi seçer. İki parametresinden ilki string'den seçilecek alt-stringin başlangıç yerini, ikinci parametre ise seçilecekstringin uzunluğunu belirtir. İkinci parametre yazılmazsa string'in sonuna kadar seçilir.

Aşağıdaki program *"Çok kısa zamanda C# dili öğrenilir."* stringinden *"C# dili"* alt-stringini seçer.

Substring01.cs

```
using System;

class Substring01
{
    public static void Main()
    {
        string str1 =  "Çok kısa zamanda C# dili öğrenilir.";
        Console.WriteLine("str1 = " + str1);
        string altString = str1.Substring(17,7);
        Console.WriteLine("str1.Substring(17,7) = " + altString);
    }
}
```

Alıştırmalar

1. Aşağıdaki program if-else yapısıyla iki metni mukayese eder. Programı satır satır çözümleyiniz.

String02.cs

```
using System;

namespace Stringler
{
    public class String02
    {
        public static void Main(string[] args)
        {
            string p = "Ankara";
            string q = "ankara";
            if (p == q)         // farklıdır
            {
                Console.WriteLine("Aynı");
            }
            else
            {
                Console.WriteLine("Farklı");
            }
        }
    }
}
```

Çıktı

```
Farklı
```

2. Aynı mukayeseyi **`public static int Compare(`**string `a` **`,`** string *`b`* **`)`** metodu ile daha kolay yapabiliriz. Bu metot `a` ve `b` yi karşılaştırır. Sözlük sıralamasına göre `a < b` ise negatif bir değer, `a ==b` ise 0 değerini, `a > b` ise pozitif bir değer alır.

   Aşağıdaki Programı satır satır çözümleyiniz.

String03.cs

```
using System;

namespace Stringler
{
    public class String01
    {
        public static void Main(string[] args)
        {
            string p = "Ankara";
            string q = "ankara";
            Console.WriteLine(String.Compare(p, q)); // çıktıis "1"
        }
    }
}
```

Sözlük sıralamasına göre "Ankara" > "ankara" olduğu için Compare(p,q) metodu "1" değerini almaktadır.

3. Aşağıdaki Programı satır satır çözümleyiniz.

IndexOf.cs

```
using System;

class IndexOf
{

    public static void Main()
    {
        string[] myStrings = { "To", "be", "or", "not", "to", "be" };
        string myString = String.Join(".", myStrings);

        char[] myChars = { 'b', 'e' };
        int index = myString.IndexOfAny(myChars);
        Console.WriteLine("'b' and 'e' occur at index " + index + " of
myString");
        index = myString.LastIndexOfAny(myChars);
        Console.WriteLine("'b' and 'e' last occur at index " + index + "
of myString");

    }

}
```

4. Aşağıdaki Programı satır satır çözümleyiniz.

```
using System;

class MainClass
{
    public static void Main()
    {
        string[] myStrings = { "To", "be", "or", "not", "to", "be" };
        string myString = String.Join(".", myStrings);

        string myString21 = myString.Substring(3);
        Console.WriteLine("myString.Substring(3) = " + myString21);
        string myString22 = myString.Substring(3, 2);
        Console.WriteLine("myString.Substring(3, 2) = " + myString22);
    }
}
```

5. Aşağıdaki deyimlerin sonunda `str` stringinin değeri nedir?

```
string str = "Sevimli küçük tavşan";
str += " tembel karabaşın üstünden atladı; ";
str += " ama karabaş farketmedi.";
```

6. Aşağıdaki kodlar çalışınca `str` ne olur?

```
string str = "Merhaba";
str.Replace ("m", "M");
```

7. Aşağıdaki kodlar çalışınca str ne olur?

```
string str = "Merhaba";
str.Replace ("M", "m");
```

8. Aşağıdaki kodlar çalışınca `sonuç` ne olur?

```
string kitap = "C# ile Nesne Programlama";
int sonuç;

sonuç = ad.Length;
sonuç = "Nesne Programlama".Length;
```

9. Aşağıdaki deyimlerin karşılarına sonuçlarını yazınız.

```
string str = "Sevimli küçük tavşan tembel karabaşın üstünden atladı";
int sonuç;

sonuç = str.IndexOf("küçük");           // sonuç :
sonuç = str.LastIndexOf("karabaş");     // sonuç :
sonuç = str.IndexOf("tembel");          // sonuç :
sonuç = str.LastIndexOf("üstü");        // sonuç :
```

10. Aşağıdaki deyimlerin karşılarına sonuçlarını yazınız.

```
string str = "Sevimli küçük tavşan tembel karabaşın sırtından atladı";
int sonuç;

sonuç = str.IndexOf("an");              // sonuç :
sonuç = str.IndexOf("an", 1);           // sonuç :
sonuç = str.IndexOf("an", 2);           // sonuç :
sonuç = str.LastIndexOf("an");          // sonuç :
sonuç = str.LastIndexOf("an",33);       // sonuç :
sonuç = str.LastIndexOf("an",32);       // sonuç :
```

11. Aşağıdaki deyimlerin karşılarına sonuçlarını yazınız.

```
string str = "Sevimli küçük tavşan tembel karabaşın sırtından atladı";
int sonuç;

sonuç = str.IndexOf("küçük");           // sonuç :
sonuç = str.IndexOf("tembel", 5, 4);    // sonuç :
sonuç = str.LastIndexOf("at");          // sonuç :
sonuç = str.LastIndexOf("sır", 25, 5);  // sonuç :
```

12. Aşağıdaki deyimlerin karşılarına sonuçlarını yazınız.

```
string str = "Sevimli küçük tavşan tembel karabaşın sırtından atladı";
bool sonuç;

sonuç = str.StartsWith("tembel karabaş"); // sonuç :
sonuç = str.StartsWith("sırtından");      // sonuç :
sonuç = str.EndsWith("Sevimli");          // sonuç :
sonuç = str.EndsWith("küçük tavşan");     // sonuç :
```

13. "Sevimli küçük tavşan tembel karabaşın sırtından atladı";

```
string str = "Sevimli küçük tavşan tembel karabaşın sırtından atladı";
bool sonuç;

sonuç = str.Contains("küçük tavşan");      // sonuç :
sonuç = str.Contains("tembel köpek");      // sonuç :
```

14. `String` ile `string` arasında fark olup olmadığını gösteren aşağıdaki programı çözümleyiniz.

```
using System;

class Uygulama
{
    public static void Main()
    {
        string s = "a";
        String S = "A";

        if (s.GetType() == S.GetType())
        {
        Console.WriteLine("string ve String aynı tiptir")
else
Console.WriteLine("string ve String aynı tip değildir");
        }
    }
}
```

15. Aşağıdaki programın kodlarını çıktı ile karşılaştırarak çözümleyiniz.

```
using System;

namespace Test
{
    class Program
    {
        static void Main(string[] args)
        {
            string test1 = "Merhaba";
            string test2 = "Dünya";

           System.Console.WriteLine(test1 + " " + test2);
            System.Console.WriteLine("Test1[0]: " + test1[0]);

            string[] test3 = new string[2];
            test3[0] = test1;
            test3[1] = test2;

            System.Console.WriteLine("Test3[0]: " + test3[0]);
            System.Console.WriteLine("Test3[1]: " + test3[1]);
            System.Console.WriteLine("Test3[0][0]: " + test3[0][0]);
        }
    }
}
```

Çıktı

```
Merhaba Dünya
Test1[0]: M
Test3[0]: Merhaba
Test3[1]: Dünya
Test3[0][0]: M
```

Bölüm 16

Char Yapısı

Char Yapısı

C# dilinde `Char` bir sınıf değil, bir yapıdır (struct). `Unicode` karakterlerini temsil eder.

Yapı ile sınıf arasındaki ayrımın kullanıcı için önem taşımadığı yerlerde, `yapı` sözcüğü yerine sınıf sözcüğünü kullanacağımızı söylemiştik. Böyle yapmakla, sistematik yapıdan bir kayba uğramayız ama pedagojik açıdan biraz daha avantajlı olacağız. Çünkü başlangıçta öğrenme sürecinde fazla ayrıntıya girmemiş olacağız. Ama genellikten bir şey yitirmeyeceğiz. Çünkü, yapı ile ilgili işleri sınıf ile yapabiliriz.

Bilgisayarlar Alfabe Bilmez

İlkönce şunu bilmeliyiz. Bilgisayarlar sayıları, harfleri ve öteki simgeleri belleklerinde bizim gördüğümüz gibi tutmazlar. Onlar her şeyi ikili sayıtlama dizgesine (binary system) göre belleklerinde tutarlar. Örneğin, 'A' harfinin ASCII kodu 65 dir. Bunun ikili sayıtlama dizgesindeki karşılığı 1000001 dir. Dolayısıyla 'A' harfi için bellekte tutulan budur. Benzer olarak, '7' rakamımın ASCII kodu 55 dir. Bunun ikili sayıtlama dizgesindeki karşılığı 110111 dir. Dolayısıyla '7' rakamı için bellekte tutulan budur.

Öyleyse, bilgisayara 'A' harfini veya '7' rakamını girdi olarak verdiğimizde, sistem onu otomatik olarak ikili sistemdeki karşılığına çevirir. Bunun tersi de doğrudur. Bilgisayar belleğindeki 10000001 dizisi bize çıktı olarak gelirken 'A' simgesine dönüşür. Aslında ekranda veya printerde bize görünen 'A' çıktısı bir resimdir; 10000001 dizisinin grafiğe dönüşmüş bir temsilidir. Bu temsili latin alfabesinde, arap alfabesinde veya japon alfabesinde farklı simgelerle gösterebilirsiniz. Girdi/çıktı işlemlerinde daima bu dönüşümler

olmaktadır. Bilgisayardan karakter çıktıları bize (ekrana veya printere) birer grafik olarak geldiğine göre, çıktıyı her alfabe için istenen biçime dönüştürmek mümkündür. Bu dönüşümü işletim sistemi otomatik yapar. Programcı olarak işin o yanıyla ilgilenmeyiz. Ama o grafiklerin (karakter) çıktıda nereye yerleşeceğini ayarlayabiliriz. Genel olarak, çıktının biçemi (format) denilen bu iş, metni (string) veya sayıyı oluşturan karakterlerin istenen sıra ve biçemde yazılmasından ibarettir. Aslında string'in kendisi bir biçemdir. Bilgisayardan çıktılar bize bir string olarak ulaşır. Şimdi C# dilinde çıktının nasıl biçemlendiğini örnekler üzerinde açıklayacağız.

Unicode Sistemi

Bilgisayarın tarihi gelişimine bakarsak, karakterler önceleri 7 sonra 8 bitlik sistemle temsil edildi. 7 bit ile yazılabilecek farklı karakter sayısı 128 , 8 bit ile yazılabilecek farklı karakter sayısı 256 dır. En yaygın kullanılan ASCII kodlama sisteminde önce karakterler 7 bit ile temsil edildi. Her karaktere 0 – 127 arasında bir numara verildi. Sonra 8 bitlik genişletilmiş ASCII kodlama sistemi kullanıldı. Bir bit'in eklenerek 7 bit'lik sistemden `8 bit`'lik sisteme geçilmesi, kullanılabilecek karakter sayısını 128 den 256 ya çıkardı. Ancak, 256 karakter dünyadaki bütün alfabelere yetmedi. O nedenle, uzun süre bilgisayar dünyası Latin alfabesini kullandı. Daha sonra, piyasa talebi, farklı ülke ve kültürlere hitap etme gereğini ortaya çıkardı. Karakterleri 16 bitlik adreste tutan `unicode` sistemine geçildi. ASCII kodlama sistemindeki gibi kullanılırsa, 16 bitlik sistemde 65536 farklı karakter temsil edilebilir. Bu dünyadaki bütün alfabeler için yeterli sanılabilir. Ama yetmez. Çünkü, yalnızca Çin alfabesinde 80.000 den fazla karakter (sembol) vardır. O zaman 16 bitlik değil, 32 bitlik sistemin kullanılması bir çözüm olarak görülebilir. 32 bitlik kodlama sistemi kurulsaydı 4 294 967 295 farklı karakter temsil edilebilirdi. Ama, her karaktere 32 bitlik yer ayıran bir sistemin öğrenilmesi çok zor olacağı gibi, bellekte çok büyük yer kaplayacağı da açıktır. O nedenle, akıllıca bir yol düşünüldü. 16 bitlik sistemde, önce *taban* (*base*) olan bir grup karakter yaratıldı. Öteki karakterler bu taban karakterlerin birleşimi olarak tanımlandı. Böylece, dünyadaki bütün alfabeleri 16 bitlik sistemle temsil edebilme olanağı yaratıldı. Java, C# ve benzeri çağdaş diller `unicode` sistemini kullanır. O nedenledir ki, biz bu kitapta değişken, fonksiyon, sınıf adlandırmalarında çekinmeden ç,Ç, İ,ı, ğ,Ğ, ö,Ö, ü,Ü, ş,Ş harflerini kullanacağız. Bilindiği gibi, 8 bitlik sistemi kullanan bazı dillerde, bu harflerin ad *(identifier)* içinde kullanılması sorunlar yaratmaktadır.

Aşağıdaki program, kullanıcının klavyeden gireceği bir karakterin unicode numarasını 10 tabanlı ve 16 tabanlı sayıtlama sistemlerinde göstermektedir.

UnicodeYaz01.cs

```
using System;

namespace Methods
{
    class UnicodeYaz01
    {
        static void Main()
        {
            Console.WriteLine("Hangi harfin unicodu'nu istiyorsunuz?");
            int kod = Console.Read();
            Console.WriteLine("{0} \t {1} \t U+{2:x4}", (char)kod,
(int)kod, (int)kod);
            }
        }
    }
```

Çıktı

```
Hangi harfin unicodu'nu istiyorsunuz?
Ğ
```

```
Ğ     286    U+011e
```

Bu çıktıyı açıklamak yararlı olabilir.

Console.Read() metodu kullanıcının kalavyeden girdiği karakteri okur. Buffer'a karakterin unicode numarası binary olarak girer.

kod = Console.Read() atama deyimi kapalı (implicit) dönüşüm yapar ve buffer'daki binary değeri int tipine çevirir.

Son satırdaki {0}, {1} ve {2} yer tutucudur. Sırasıyla, (char)kod, (int)kod, (int)kod) değişkenlerinin çıktıda yazılacağı yerleri belirlerler.

\t simgesi tab yazar, çıktıda aynı satıra yazılan ardışık değerler birbirinden bir tab kadar uzağa yerleşir.

U+{2:x4} ifadesinde U+ simgesi unicode'u gösteren bir öntakıdır, string olarak çıktıya olduğu gibi yazılır. {2:x4} simgesinde (2) yer tutucudur, 2-inci değişkenin çıktıdaki yerini belirler. (:) çıktıda biçemin başlama yeridir. (x) çıktının hexadecimal (16 tabanlı sistem) yazılmasını sağlar. (4) çıktının 4 haneye sağa yanaşık yazılmasını sağlar.

Aşağıdaki program Türk alfabesindeki büyük ve küçük harflerin karşısına `unicode` ve `hexadecimal` (16 tabanlı sayıtlama sistemi) numaralarını yazdırır.

UnicodeYaz02.cs

```
using System;

namespace Methods
{
    class UnicodeYaz02
    {
        static void Main()
        {
            string text =
"AaBbCcÇçDdEeFfGgĞğHhIıİiJjKkLlMmNnOoÖöPpRrSsŞşTtUuÜüVvYyZz";
            foreach (char c in text)
            {
                Console.Write("{0} \t {1} \t U+{2:x4}", c, (int)c,
(int)c);
                Console.WriteLine("{0} \t {1} \t U+{2:x4}", c, (int)c,
(int)c);
            }
        }
    }
}
```

Bu programın çıktısını bir tablo biçiminde gösterirsek, Türkçe alfabenin unicode numaralarını görüyor olacağız. Gerektiğinde bu tabloyu referans olarak kullanabiliriz.

Türkçe Alfabenin Unicode Kodları

Harf	Unicode	Hex	Harf	Unicode	Hex
A	65	U+0041	a	97	U+0061
B	66	U+0042	b	98	U+0062
C	67	U+0043	c	99	U+0063

Ç	199	U+00c7	ç	231	U+00e7
D	68	U+0044	d	100	U+0064
E	69	U+0045	e	101	U+0065
F	70	U+0046	f	102	U+0066
G	71	U+0047	g	103	U+0067
Ğ	286	U+011e	ğ	287	U+011f
H	72	U+0048	h	104	U+0068
I	73	U+0049	ı	305	U+0131
İ	304	U+0130	i	105	U+0069
J	74	U+004a	j	106	U+006a
K	75	U+004b	k	107	U+006b
L	76	U+004c	l	108	U+006c
M	77	U+004d	m	109	U+006d
N	78	U+004e	n	110	U+006e
O	79	U+004f	o	111	U+006f
Ö	214	U+00d6	ö	246	U+00f6
P	80	U+0050	p	112	U+0070
R	82	U+0052	r	114	U+0072
S	83	U+0053	s	115	U+0073
Ş	350	U+015e	ş	351	U+015f
T	84	U+0054	t	116	U+0074
U	85	U+0055	u	117	U+0075
Ü	220	U+00dc	ü	252	U+00fc

V	86	U+0056	v	118	U+0076
Y	89	U+0059	y	121	U+0079
Z	90	U+005a	z	122	U+007a

Bu tabloya dikkat edersek, İngiliz alfabesinde de olan harflerimiz unicode numarası olarak ASCII kodlarını almaktadır. Ama Ç, ç, Ö, ö, Ü, ü harflerinin kodları 127 bariyerini aşmaktadır, ama 255 den küçüktür. Dolayısıyla onlar genişletilmiş ASCII karakterleri sınıfındadır. Böyle olması doğaldır, çünkü bu harfler bazı avrupa ülkelerinin alfabelerinde de vardır ve genişletilmiş ASCII kümesine dahil edilmişlerdir. Öte yandan, avrupa alfabelerinde yer almayan Ğ, ğ, ı, İ, Ş, ş harflerinin unicode numaraları 255 bariyerini aşmaktadır. Onlar genişletilmiş ASCII kümesine ait değildirler.

Ayrıca şuna dikkat etmeliyiz. Alfabedeki sıralama kodlara da yansımıştır; alfabetik sıralamada önce gelen harfin kodu sonra gelen harfin kodundan daha küçüktür. Bu olgu çok önemlidir. Bilgisayarla yapılan alfabetik sıralamanın alıştığımız sırada olmasını sağlar. Çünkü alfabetik ya da daha genel olarak string sıralamaları karakterlerin unicode numaralarına göre yapılır.

Yukarıdaki programda

```
text="0123456789"
```

koyarsak, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 rakamlarının unicode ve hexadecimal kodlarını elde ederiz.

Rakam	**Unicode**	**Hex**
1	49	U+0031
2	50	U+0032
3	51	U+0033
4	52	U+0034
5	53	U+0035
6	54	U+0036
7	55	U+0037
8	56	U+0038
9	57	U+0039

Char Yapısı

Bu yapının unicode karakterlerini temsil eden bir sınıf olduğunu söylemiştik. Bu bölümde, `Char` yapısının sistematik incelemesine girmeden onun başlıca öğelerini ve ne işe yaradıklarını açıklayacağız.

Veri Alanları

İki tanedir: `MaxValue, MinValue.`

`MaxValue` temsil edilen karakter kodlarının en büyüğünü gösterir. Hexadecimal değeri 0xFFFF olan sabit sayıdır.

`MinValue` temsil edilen karakter kodlarının en küçüğünü gösterir. Hexadecimal değeri 0xFFFF olan sabit sayıdır. Hexadecimal değeri 0x00 olan sabit sayıdır.

Metotlar

Karakterlerin mukayese edilmesi, kodlarının bulunması, harf, sayı, sembol olup olmadıklarının saptanması, büyük ya da küçük harf karşılığına dönüştürülmesi, standart çıktıya string olarak yazılması gibi işleri yapan metotlar şunlardır.

CompareTo , Equals , GetHashCode , GetNumericValue , GetType , GetTypeCode , GetUnicodeCategory , IsControl , IsDigit , IsLetter , IsLetterOrDigit , IsLower , IsNumber , IsPunctuation , IsSeparator , IsSurrogate , IsSymbol , IsUpper , IsWhiteSpace , Parse , ToLower , ToString , ToUpper.

Bu metotların adları, her birinin işlevi hakkında yeterli ipucu vermektedirler. O nedenle, bir kaçıyla ilgili uygulamalar yapmakla yetineceğiz.

Char.GetNumericValue() Metodu:

Bir karakter 0,1,2,3,4,5,6,7,8,9 rakamlarından biri ise, o karakterin sayısal değerini; değilse -1 değerini alır. İki biçimi vardır:

```
public static double GetNumericValue(char c);
public static double GetNumericValue(string s,int index);
```

Örneğin,

```
GetNumericValue('3')==3
GetNumericValue('a')==-1
```

olur. Bu metot bir string içinde sıra sayısı (index) verilen bir karakter için benzer işi yapar. Örneğin,

```
GetNumericValue("abc75dfg",4))==  5
GetNumericValue("abc75dfg",6))== -1
```

olur. Çünkü birinci satırda "abc75dfg" stringindeki 4-üncü karakter 5 rakamıdır; metot 5 değerini verir. İkinci satırda 6-ıncı karakter ise rakam değil 'f' harfidir; metot -1 değerini verir.

Bu dört satırı bir program içine koyarak, çıktıları görebiliriz.

```
using System;

namespace metotlar
{
    class Char01
    {
        public static void Main()
        {
            Console.WriteLine(Char.GetNumericValue('3'));    // Çıktı:  3
            Console.WriteLine(Char.GetNumericValue('a'))   ; // Çıktı: -1
```

```
            Console.WriteLine(Char.GetNumericValue("abc75dfg", 4));  //
Çıktı:  5
            Console.WriteLine(Char.GetNumericValue("abc75dfg", 6));  //
Çıktı:  -1
        }
    }
}
```

Char.IsLetter() Metodu

Bir karakterin ya da bir string içinde sırası verilen karakterin bir harf olup olmadığını belirler. Harf ise `True` değeri, değilse `False` değeri alır.

Char02.cs

```
using System;

public class Char02
{
    public static void Main()
    {
        Console.WriteLine(Char.IsLetter('7'));              // False
        Console.WriteLine(Char.IsLetter("Başkent", 3));     // True
    }
}
```

Char.ToUpper() Metodu

Bir unicode karakteri büyük harfe dönüştürür. Tabii, dönüşecek karakterin, ilgili alfabede büyük harf karşılığının olması gerekir. Metodun iki farklı biçimi vardır. Birincisi işletim sisteminin ait olduğu dil/kültür threadinde çalışır. Diğeri ise, işletim sisteminin ait olduğu dil/kültür threadinden farklı bir thread için çalışır. Aşağıdaki örnek, her iki metodon işlevini göstermektedir.

```
public static char ToUpper(char c);
public static char ToUpper(char c,CultureInfo culture);
```

Char03.cs

```
using System;
using System.Globalization;
using System.Threading;
namespace Metotlar
{
    public class IsLetterSample
    {
        public static void Main()
        {
            Console.WriteLine(Char.ToUpper('a'));                  // A
            Console.WriteLine(Char.ToUpper('B'));                  // B
            Console.WriteLine(Char.ToUpper('w', new CultureInfo("de-
DE"))); // W
            Console.WriteLine(Char.ToUpper('W', new CultureInfo("de-
DE"))); // W
        }
    }
}
```

Aşağıdaki örnek Char sınıfının başka fonksiyonlarının kullanılışını göstermektedir.

Char04.cs

```
using System;

public class CharStructureSample
{
    public static void Main()
    {
        char chA = 'A';
        char ch1 = '1';
        string str = "test string";

        Console.WriteLine(chA.CompareTo('B'));          // Çıktı: "-1"
('A' harfi 'B' harfinden önce gelir demektir.)
        Console.WriteLine('B'.CompareTo('A'));          // Çıktı: "1"
('B' harfi 'A' harfinden önce gelir demektir.)
        Console.WriteLine(chA.Equals('A'));             // Çıktı: "True"
        Console.WriteLine(Char.GetNumericValue(ch1));   // Çıktı: "1"
        Console.WriteLine(Char.IsControl('\t'));        // Çıktı: "True"
        Console.WriteLine(Char.IsDigit(ch1));           // Çıktı: "True"
        Console.WriteLine(Char.IsLetter(','));          // Çıktı: "False"
        Console.WriteLine(Char.IsLower('u'));           // Çıktı: "True"
        Console.WriteLine(Char.IsNumber(ch1));          // Çıktı: "True"
        Console.WriteLine(Char.IsPunctuation('.'));     // Çıktı: "True"
        Console.WriteLine(Char.IsSeparator(str, 4));    // Çıktı: "True"
        Console.WriteLine(Char.IsSymbol('+'));          // Çıktı: "True"
        Console.WriteLine(Char.IsWhiteSpace(str, 4));   // Çıktı: "True"
        Console.WriteLine(Char.Parse("S"));             // Çıktı: "S"
        Console.WriteLine(Char.ToLower('M'));           // Çıktı: "m"
        Console.WriteLine('x'.ToString());              // Çıktı: "x"
    }
}
```

```
using System;

public class CharDemo
{
    public static void Main()
    {
        string str = "This is a test. $23";
        int i;

        for (i = 0; i < str.Length; i++)
        {
            Console.Write(str[i] + " is");
            if (Char.IsDigit(str[i]))
                Console.Write(" digit");
            if (Char.IsLetter(str[i]))
                Console.Write(" letter");
            if (Char.IsLower(str[i]))
                Console.Write(" lowercase");
            if (Char.IsUpper(str[i]))
                Console.Write(" uppercase");
            if (Char.IsSymbol(str[i]))
                Console.Write(" symbol");
            if (Char.IsSeparator(str[i]))
```

```
                Console.Write(" separator");
            if (Char.IsWhiteSpace(str[i]))
                Console.Write(" whitespace");
            if (Char.IsPunctuation(str[i]))
                Console.Write(" punctuation");

            Console.WriteLine();
        }

        Console.WriteLine("Original: " + str);

        // Convert to upper case.
        string newstr = "";
        for (i = 0; i < str.Length; i++)
            newstr += Char.ToUpper(str[i]);

        Console.WriteLine("Uppercased: " + newstr);
    }
}
```

Alıştırmalar

1. İki stringin ilk harflerinin aynı olup olmadığını bulan aşağıdaki programı çözümleyiniz. Programı değiştirerek, stringleri kullanıcının girmesini sağlayınız.

```
using System;

class Test
{
    string s1 = "Merhaba";
    string s2 = "Dünya";
    static void Main(string[] args)
    {
        Test t = new Test();
        if (t.s1[0] == t.s2[0])
            Console.WriteLine("İlk harfler aynıdır");
        else
            Console.WriteLine("İlk harfler farklıdır");
    }
}
```

2. Aşağıdaki program bir tümcedeki sözcüklerin sırasını ters çevirmektedir. Programı çözümleyiniz.

```
using System;
public static class TersKelimeler
{
    public static string TersKelime(this string str, char ayrac)
    {
        char yedek;
        int sol = 0, orta = 0;

        char[] chars = str.ToCharArray();
```

```
        Array.Reverse(chars);

        for (int i = 0; i <= chars.Length; i++)
        {
            if (i != chars.Length && chars[i] != ayrac)
                continue;

            if (sol == i || sol + 1 == i)
            {
                sol = i + 1;
                continue;
            }

            orta = (i - sol - 1) / 2 + sol;

            for (int j = i - 1; j > orta; j--, sol++)
            {
                yedek = chars[sol];
                chars[sol] = chars[j];
                chars[j] = yedek;
            }

            sol = i + 1;
        }

        return new String(chars);
    }

    public static string TersKelime(this string str)
    { return str.TersKelime(' ');
    }
    public static void Main()
    {
        string str = "Yüzyılın cahilleri kimler?";

        string strTers = TersKelimeler.TersKelime(str);
        Console.WriteLine(str);
        Console.WriteLine(strTers);
    }
}
```

3. Bir stringdeki karakterlerin sırasını ters çeviren bir program yazınız.

Bölüm 17

Kalıtım (inheritance)

Kalıtım Kavramı

Mesaj İletme

Polimorfizm

Kalıtım kavramı nesne yönelimli programlamanın üç önemli kavramından birisidir. Öteki ikisi *polymorphism* ve *interface*'dir. Bu bölümde kalıtım kavramının ne olduğunu açıklayacağız. Bu kavramın çok önemli uygulamaları, bu kitabın kapsamı dışındadır.

Adından da anlaşılacağı üzere, en basit ifadesiyle kalıtım, atadan oğula bırakılandır. Bunu nesne yönelimli programlama için ifade edersek şöyle diyebiliriz: Kalıtım, bir sınıftan başka bir sınıfa miras bırakılandır. Bunu aşağıdaki basit örnek üzerinde açıklamak, algılanmasını kolaylaştıracaktır.

Kalıtım01.cs

```
using System;

public class AtaSınıf
{
    public AtaSınıf()
    {
        Console.WriteLine("Ata Kurucu.");
    }

    public void Yaz()
    {
        Console.WriteLine("Ben Ata sınıfındayım.");
    }
}

public class OğulSınıf : AtaSınıf
{
    public OğulSınıf()
```

```
    {
        Console.WriteLine("Oğul Kurucu.");
    }
}

class Uygulama
{
    public static void Main()
    {
        OğulSınıf oğul = new OğulSınıf();

        oğul.Yaz();
    }
}
```

Şimdi bu programı çözümleyelim. `AtaSınıf` içinde `public AtaSınıf()` kurucusu çağrılınca konsola "Ata Kurucu" metni yazılıyor. Ayrıca `AtaSınıf` içinde `Yaz()` metodu "Ben Ata sınıfındayım." metnini konsola yazıyor.

`OğulSınıf` bildirimi

```
public class OğulSınıf : AtaSınıf
```

sözdizimi ile veriliyor. Burada alışılmışın dışında

```
OğulSınıf : AtaSınıf
```

dizgesi var. Bu özel bir sözdizimidir (syntax) ve anlamı şudur: `OğulSınıf` sınıfının `AtaSınıf` sınıfından miras aldığını belirtir. `OğulSınıf : AtaSınıf` dizgesinde `OğulSınıf` miras alan, `AtaSınıf` ise miras bırakandır. İkisi arasına (:) konulması zorunludur. `OğulSınıf` içinde `public OğulSınıf()` kurucusu çağrılınca konsola "Oğul Kurucu" metni yazılıyor.

Şimdi bu iki sınıfı çağıran kodları çalıştıracak bir `Main()` metodu ve onu içerecek bir sınıf gerekiyor. `Uygulama` sınıfı bu işe yarıyor. `Uygulama` sınıfı içindeki `Main()` metodu `OğulSınıf` sınıfa ait `oğul` adlı bir nesne yaratıyor ve onun `Yaz()` metodunu çağırıyor. Oysa, `Yaz()` metodu `OğulSınıf` sınıfa değil, `AtaSınıf` 'a aittir. Ama `OğulSınıf` 'a miras kaldığı için, onu kendi öğesiymiş gibi kullanır. Böylece,

```
oğul.Yaz();
```

deyimi `AtaSınıf` içindeki `Yaz()` metodunu çağırır ve "Ben Ata sınıfındayım." metnini konsola yazdırır.

Miras bırakan sınıfa *taban sınıf* (base class), miras alan sınıfa *türemiş sınıf* (inherited class) denilir. Bunlara eşanlamlı olarak, taban sınıfa `ata`, türemiş sınıfa da `oğul` diyeceğiz.

Mesaj İletme

Uygulama sınıfından `oğul.Yaz()` metodunu çağırdık. Buna, nesne yönelimli programlamada bir sınıftan başka bir sınıfa *mesaj gönderme* denilir. Mesaj gönderme, aslında başka bir sınıftaki metodu çağırma; yani onu harekete geçirme eylemidir. Büyük bir programda, farklı eylemler farklı sınıflardaki metotlarla yapılır. Gerektiğinde, bir sınıftan başka bir sınıfın metodu çağrılıp, işlevini yapması istenebilir.

Aşağıdaki örnek, kullanıcıdan aldığı veriyi hem oğul (türemiş) sınıfındaki `apartmanYöneticisi`

alanına, hem de oğul sınıfı üzerinden ata sınıfındaki (taban sınıf) `evSahibi` alanına taşımaktadır.

Kalıtım02.cs

```
using System;

class Ev
{
    public string sahip;
    public Ev()
    {
        Console.WriteLine("Ev Kurucu.");
    }

    public void EvSahibiniYaz(string sahip)
    {
        Console.WriteLine("Ev 'in sahibi : {0}", sahip);

    }
}

class ApartmanKatı : Ev
{
    string apartmanYöneticisi;
    ApartmanKatı()
    {
        Console.WriteLine("ApartmanKatı Kurucu.");
    }
    void AptYöneticiniYaz(string s)
    {
        Console.WriteLine("Apartman Yöneticis : {0}", s);
    }

    class Uygulama
    {
        public static void Main()
        {
            ApartmanKatı apt = new ApartmanKatı();

            Console.WriteLine("Evin sahibi kim?  ");
            apt.sahip = Console.ReadLine();
            apt.EvSahibiniYaz(apt.sahip);

            Console.WriteLine("Apartman Yöneticisi kim?  ");
            apt.apartmanYöneticisi = Console.ReadLine();
            apt.AptYöneticiniYaz(apt.apartmanYöneticisi);
        }
    }
}
```

Burada `Ev` sınıfı, bütün evleri temsil eden genel bir sınıftır. `ApartmanKatı` ise `Ev` sınıfının bazı niteliklerini taşır, ama kendine özgü niteliklere de sahiptir. Örneğin, `Ev` sınıfında *salon, mutfak, oda sayısı, ısıtma sistemi, emlakVergisi, fiyatı* gibi değişken ve metotlar varsa, onlar `apartmanKatı` için de geçerli olur. Ama `apartmanKatında` her evde olması gerekmeyen bazı nitelikler var olabilir. Örneğin, *evin kaçıncı katta olduğu, apartman yöneticisi, kapıcı, kalorferci* gibi nitelikler.

Bu söylediklerimizin yukarıdaki `Ev` ve `apartmanKatı` sınıflarının gövdelerine yazabiliriz. Ancak, görünüş ne denli yalın olursa, konuyu kavramamız o kadar kolaylaşacaktır. O nedenle, yukarıdaki örnekte

ata sınıfa `evSahibi` adlı alan ile `EvSahibiniYaz()` metodunu koyduk. Benzer olarak oğula `apartmanYöneticisi` alanı ile `ApartmanYöneticisiniYaz()` metodunu koyduk. Alanlar ve metotlar, uygulamanın gerektirdiği kadar artırılabilir.

Şimddi yukarıdaki sözdizimini açıklayalım:

Kalıtım03.cs

```
class Ev
{
    public string evSahibi;
    public Ev()
    {
        Console.WriteLine("Ev Kurucu.");
    }

    public void EvSahibiniYaz (string evSahibi)
    {
        Console.WriteLine("Ev 'in sahibi : {0}", evSahibi);

    }
}
```

bildirimi `Ev` adlı ata sınıfın bildirimidir. Bunun gövdesindeki bütün öğeler `public` erişim belirteci ile nitelenmiştir. Oğuldan (alt sınıftan) bunlara erişilebilmesi için `public` nitelemesi gereklidir.

`public string evSahibi` bir veri alanıdır (field).

`public Ev() { }` deyimi Ev sınıfının kurucusudur.

`public void EvSahibiniYaz (string evSahibi) {}` deyimi bir metot bildirimidir.

Aynı düşünüşle,

Kalıtım04.cs

```
class ApartmanKatı : Ev
{
    string apartmanYöneticisi;
    public ApartmanKatı()
    {
        Console.WriteLine("ApartmanKatı Kurucu.");
    }
    public void AptYöneticiniYaz(string s)
    {
        Console.WriteLine("Apartman Yöneticis : {0}", s);
    }
}
```

bildirimi `apartmanKatı` adlı oğul sınıfının bildirimidir. Bu bildirimin başlığındaki

```
public class apartmanKatı : Ev
```

ifadesi, `apartmanKatı` sınıfının `Ev` sınıfından miras aldığını belirtir. `apartmanKatı : Ev` dizgesinde `apartmanKatı` miras alan (oğul), `Ev` ise miras bırakandır (ata). İkisi arasına (:) konulması zorunludur.

Oğulun gövdesindeki öğelere herhangi bir erişim belirteci nitelemesi konulmamıştır. Öyleyse hepsi `private` nitelemesine sahiptir. Böyle olduğu halde, erişimde bir sorun yaşanmaz. Çünkü, `Uygulama` sınıfındaki `Main()` metodu bu sınıfa ait `apt` adlı bir nesne yaratmaktadır. `Apt` nesnesi içinden `apt`

nesnesinin öğelerine kısıtsız erişmek mümkündür; bir erişim belirtecine gerek yoktur.

`string apartmanYöneticisi` oğul sınıfında bir veri alanıdır.

`public apartmanKatı() { }` deyimi oğul sınıfının kurucudur.

`public void` AptYöneticiniYaz(`string` s) oğul sınıfında bir metot bildirimidir.

Oğul sınıf, ata sınıfın `private` olmayan bütün öğelerine (alanlar ve metotlar) sahip olur. Ayrıca kendisine özgü öğeler olabilir. Oğul sınıftan ata sınıfa doğal dönüşüm (casting, implicit conversion) yapılabilir. Ters dönüşüm için apaçık (explicit) dönüşüm gerekir. Örneğin, yukarıdaki ata ve oğul için, şu dönüşümler geçerlidir.

```
Ev birEv = new Ev();                              ataya ait nesne yarat
ApartmanKatı birAptKatı = new ApartmanKatı();     oğula ait nesne yarat
birEv = birAptKatı;                               ataya atama
birAptKatı = (ApartmanKatı)birEv;                 oğula atama
```

Bu dönüşümlerde ata ve oğuldaki ortak öğelerde bir kayıp olmaz.

İpucu

Oğulsınıfa erişildiğinde ata sınıfa da erişilmiş olur; yani ata sınıfın private olmayan öğelerine oğul sınıf üzerinden erişilir.

Polimorfizm

Poly “çok” morph is “form”, “biçim” anlamlarını taşır. Bu ikisinin birleşimiyle oluşan “poymorphism” sözcüğü “çok biçimlilik” amlamına gelir.

Bunu şöyle bir örnekle açıklayalım. Çaldığında farklı marka telefonlar farklı sesler çıkarırlar. Alışılmış telefonlar bir zil çalar. Yeni kuşak telefonlar ve cep telefonları sayısız farklı sesler verebilirler. Telekom şirketi bu telefonların nasıl ses çıkardığını bilmez ve o seslerle ilgilenmez. Telekomun görevi, aranan numaraya gerekli sinyali ulaştırmaktır. Bu işi bütün telefonlar için aynı yöntemle yapar; karşıdaki telefonun nasıl ses verdiğine bakmaz. Bu işi yapan bir ata (taban, base) sınıfı olduğunu varsayalım. Bunun her telefonda farklı bir oğulu (inherited) sınıfı var. O sınıflarda ses özeliği hepsinde farklı farklı düzenlenebilir. Bu bir tür polimorfizmdir.

Şimdi bunu nesne yönelimli programlamada sınıflar için düşünelim. Bir atadan oğul yaratıldığında, bazan, miras geçen öğeler üzerinde değişiklikler yapılması gerekir. Örneğin, bir alan farklı veri tutsun veya bir adı ve değer kümesi korunan bir metot farklı iş yapsın istenebilir. Bunun için, öğenin new ile nitelenmesi yeterlidir. Aşağıdaki örnek bunun nasıl yapılacağını göstermektedir.

Kalıtım05.cs

```
public class Ata
{
    public void BirMetot() { }
    public int birAlan;
    public int birÖzgen
    {
        get { return 0; }
    }
```

```
}

public class Oğul : Ata
{
    public new void BirMetot() { }
    public new int birAlan;
    public new int birÖzgen
    {
        get { return 0; }
    }
}
```

Aşağıdaki örnek konuyu biraz daha açmaktadır.

Kalıtım06.cs

```
using System;

public class Çizici
{
    public virtual void Çiz()
    {
        Console.WriteLine("Çizici");
    }
}

public class DoğruÇiz : Çizici
{
    public override void Çiz()
    {
        Console.WriteLine("Line.");
    }
}

public class DaireÇiz : Çizici
{
    public override void Çiz()
    {
        Console.WriteLine("Circle");
    }
}

public class KareÇiz : Çizici
{
    public override void Çiz()
    {
        Console.WriteLine("Kare");
    }
}


public class Uygulama
{
    public static int Main()
    {
        Çizici[] birÇizici = new Çizici[4];

        birÇizici[0] = new DoğruÇiz();
```

```
        birÇizici[1] = new DaireÇiz();
        birÇizici[2] = new KareÇiz();
        birÇizici[3] = new Çizici();

        foreach (Çizici sayaç in birÇizici)
        {
            sayaç.Çiz();
        }

        return 0;
    }
}
```

Alıştırmalar

Aşağıdaki programın deyimlrini çıktıları ile karşılaştırarak programı çözümleyiniz.

Kalıtım07.cs

```
using System;
class Anne
{
    public Anne()
    {
        Console.WriteLine("Anne kurucu");
    }
    public void Selam()
    {
        Console.WriteLine("Anne size 'Merhaba' dedi");
    }
    public void Konuş()
    {
        Console.WriteLine("Anne konuşuyor");
    }
    public virtual void Şarkı()
    {
        Console.WriteLine("Anne şarkı söylüyor");
    }
};

class Çocuk : Anne
{
    public Çocuk()
    {
        Console.WriteLine("Çocuk kurucu");
    }
    public new void Konuş()
    {
        Console.WriteLine("Çocuk konuşuyor");
    }
    public override void Şarkı()
    {
        Console.WriteLine("Çocuk şarkı söylüyor");
    }
};
```

```
public class Uygulama
{
    public static void Main()
    {
        Anne a1 = new Anne();
        a1.Konuş();
        a1.Şarkı();
        a1.Selam();
    }
}
```

Çıktı

```
Anne kurucu
Anne konuşuyor
Anne şarkı söylüyor
Anne size 'Merhaba' dedi
```

Bölüm 18

Kapsülleme (Encapsulation)

Kavram

Nesne Yönelimli Programlamada, *kapsülleme* eylemi, verinin veya metodun başka yerden görünmeyecek biçimde üstünün sınıf içinde örtülmesi demektir. Genellikle, kapsüllenmiş nesneye soyut veri tipi (abstract data type) denilir.

Gerekseme

Aslında, kapsülleme yapmamızın nedeni her programcının yaptığı basit yanlışlarla verinin veya kodların bozulmasını önlemektir. Başka bir deyişle, duyarlı verilerimizi, onu işleyen metotlrla bir araya getirip paketleriz ve yanlışlıkla dışarıdan erişimi engelleriz.

Korumak istediğimiz verileri `public` nitelemek yerine `private` niteleriz. Private nitelenmiş veriler direkt olarak ancak iki yolla işlenebilir. Birinci yöntem alışılmış `accessor` ve `mutator` (get/set) metotlarını kullanmaktır. Öteki yöntem ise `özgen` (property) kullanmaktır. Hangisini kullanırsak kullanalım, verilerimiz güvende olacaktır.

Şimdi bu yöntemleri örnekler üzerinde görelim. Aşağıdaki örnekte, Mutator (set metodu) ve Accessor (get metodu) , sırasıyla, `private` nitelenmiş department `özgen`'ine değer atar ve atanmış değeri okurlar.

```
using System;
public class Bölüm
{
    private string bölümAdı;
```

```
    // Accessor (Getter)
    public string GetBölümAdı()
    {
        return bölümAdı;
    }
    // Mutator (Setter).
    public void SetBölümAdı(string a)
    {
        bölümAdı = a;
    }
}

class Uygulama
{
    public static void Main(string[] args)
    {
        Bölüm d = new Bölüm();
        d.SetBölümAdı("MUHASEBE");
        Console.WriteLine("Bölüm adı  :" + d.GetBölümAdı());

        //private bölümAdı'na nesne ile erişilemez
        Bölüm e = new Bölüm();
        // e.bölümAdı = "ARAŞTIRMA"; // Hata
        // Console.WriteLine("Bölüm adı  :" + e.bBölümAdı()); // Hata
    }
}
```

Bu örnekte, sınıfa ait bir nesne ile private nitelemeli `bölümAdı` alanına ulaşamayız. Ona değer atamak için `SetBölümAdı` metodunu, atanan değeri okumak için `GetBölümAdı()` metodunu kullanıyoruz. Bu iki metot accessor (getter) ve mutator (setter) adlarıyla da anılır.

Özgen (property) Kullanılarak Kapsülleme Yapma

Özgen kavramı C# ile ortaya konan yeni bir programlama kavramıdır. Şimdilik, bu özellik ancak bir kaç dil tarafından destekleniyor. Özgen'in görevi alana veri yazmak ve yazılan veriyi okumaktır. Alana başka türlü erişilemediği için, oradaki veri *kapsüllenmiş* sayılır. Yukarıda anlattığımız yöntem de bu işi iyi yapar. Ama C# bunu daha nezih bir yolla yapmaktadır. Bunu bir örnek üzerinde gösterelim.

Kapsülleme01.cs

```
using System;
public class Fakülte
{
    private string bölüm;
    public string Bölüm
    {
        get
        {
            return bölüm;
        }
        set
        {
            bölüm = value;
        }
    }
```

```
}
public class Uygulama
{
    public static void Main(string[] args)
    {
        Fakülte  d = new Fakülte();
        d.bölüm = "Matematik";
        Console.WriteLine("Fakülte :{0}", d.Bölüm);
    }
}
```

Çıktı

```
Error    1    'Fakülte.bölüm' is inaccessible due to its protection level ...
```

Görüldüğü gibi, program sınıfa ait bir özgene atanan değeri okuyamıyor; yani ona dışarıdan erişilemiyor. Bu programı çözümleyerek özgen ile kapsüllemenin yapılışını öğreneceğiz. Özgen'in iki metodu var: `Get()` ve `Set()`. `Get()` metodu alandaki veriyi getirir. `Set()` metodu alana veri yollar; yani değişkene değer atar. Bu işi yaparken, C# dilinin bir anahtar sözcüğünü kullanır: "`value`". Eğer `Set()` metodu konulmazsa, özgen *yalnız-okunur* (read-only) kılınmış olur.

Yalnız-okunur Özgen (Read Only Property)

Kapsülleme02.cs

```
using System;
public class Fakülte
{
    private string bölüm;
    public Fakülte(string str)
    {
        bölüm = str;
    }
    public string Bölüm
    {
        get
        {
            return bölüm;
        }
    }
}
public class Uygulama
{
    public static void Main(string[] args)
    {
        Fakülte obj = new Fakülte("Ekonomi");
        Console.WriteLine("Bölüm: {0}", obj.Bölüm);
        //obj.bölüm = "ist";
    }
}
```

Bu program, sınıfa ait `bölüm` adlı özgenin değerini okur; yani ona erişebilir. Ama ona değer atayamaz. Sınıfın tek parametreli bir kurucusu var:

```
    public Fakülte(string str)
    {
        bölüm = str;
```

```
    }
```

`Main()` metodu bu kurucuyu

```
Fakülte obj = new Fakülte("Ekonomi");
```

deyimi ile çağırınca `Fakülte` sınıfının `obj` adlı bir nesnesi kurulur ve

```
bölüm = "Ekonomi";
```

ataması yapılır.

```
Console.WriteLine("Bölüm: {0}", obj.Bölüm);
```

deyimi atanmış olan bu değeri konsola yazar.

```
    public string Bölüm
    {
        get
        {
            return bölüm;
        }
    }
```

blokuna bakarsak, Bölüm özgeni yalnızca `get` metodunu içeriyor. Bu metot bölüm değişkeninin değerini veriyor. `set` metodu olmadığı için, bölüme bir değer atanamaz. Gerçekten, programda etkisiz kılınan

```
//obj.bölüm = "ist";
```

deyimindeki // simgelerini kaldırıp deyimi etkili kılarsak, derleyici şu hata iletisini verecektir.

```
Error   1   'Fakülte.bölüm' is inaccessible due to its protection level ...
```

Yalnız-yazılır Özgen (Write Only Property)

Yalnız-okunur özgene benzer olarak *yalnız-yazılır özgenler* de yaratılabilir. Bunu yapmak için `get/set` metot ikilisinden yalnızca `set` metodunu kullanmak yetecektir. Aşağıdaki program o işi yapıyor.

```
using System;
public class Fakülte
{
    private string bölüm;
    public string Bölüm
    {
        set
        {
            bölüm = value;
            Console.WriteLine("Bölüm :{0}", bölüm);
        }
    }

    public class Uygulama
    {
        public static void Main(string[] args)
        {
            Fakülte d = new Fakülte();
            d.bölüm = "Biyoloji";
        }
    }
}
```

Bölüm 19

Arayüzler (interfaces)

Arayüz Nedir?
C# dilinde arayüz
Arayüz Bildirimi
Arayüz Kullanımı
Arayüzden Türetme

Arayüz Nedir?

Konuyu basitleştirmek için bir örnekle başlayalım. Aşağıdaki programı çözümleyince arayüzün işlevini ve nasıl kullanıldığını anlayacağız.

Arayüz01.cs

```
using System;

interface Idemo
{
    void Göster();
}

class SDemo : Idemo
{
    public void Göster()
    {
        Console.WriteLine("Idemo arayüzünü kullanıyorum");
    }

    public static void Main(string[] args)
    {
        SDemo d = new SDemo();
        d.Göster();
    }
}
```

Çözümleme:

```
interface Idemo
{
    void Göster();
}
```

bloku `Idemo` adlı bir arayüz bildirimidir. Arayüz bildirimi `interface` anahtar sözcüğü ile başlamış, hemen arkasından arayüzün `Idemo` adı yazılmıştır. Arayüz adının ilk harfi `'I'` ile başlamıştır. Arayüzün gövdesinde `void` değer alan `Göster()` adlı bir metot bildirimi vardır. Arayüz gövdesinde metodun yalnızca imzası (signature) yazılıdır, metodun *tanımı* yoktur. Bu metodun tanımı arayüzü kullanan (implement eden) `SDemo` sınıfı içinde yapılmışır:

```
    public void Göster()
    {
        Console.WriteLine("Idemo arayüzünü kullanıyorum");
    }
```

`Sdemo` sınıfı bildirimi başlarken yazılan

```
class SDemo : Idemo
```

deyimi, derleyiciye `SDemo` sınıfının `Idemo` arayüzünü kullanacağını (implement edeceğini) bildirir.

```
SDemo d = new SDemo();
```

deyimi Sdemo sınıfına ait d adlı bir nesne yaratır.

```
d.Göster();
```

deyimi, `Idemo` arayüzünde bildirilen ve `Sdemo` sınıfında tanımı yapılan `Göster()` metodunu çalıştırır.

An *interface* looks like a class, but has no implementation. The only thing it contains are definitions of *events*, *indexers*, *methods* and/or *properties*. The reason *interfaces* only provide definitions is because they are inherited by *classes* and *structs*, which must provide an implementation for each interface member defined.

C# dilinde `arayüz`

C# dilinde `arayüz` (interface) soyut bir veri tipidir. Sınıfa benzer şekilde bildirilir. Ancak içerdiği öğelerin tanımı değil, yalnızca imzaları vardır. Arayüzler öğe olarak şunları içerebilirler:

Özgen imzaları (signature of properties)
Metot imzaları (method signature)
Delegeler (delegates)
Olaylar (events)
İndeksçiler (indexers)

Arayüzler öğe olarak şunları içeremezler:

Alan (field)
Kurucu
Yokedici

Arayüz asla içerdiği öğeleri tanımlamaz; öyle olmasının basit bir nedeni vardır. Bir arayüz farklı sınıflar tarafından farklı amaçlarla kullanılabilir. Her sınıf, kullandığı (implement) arayüzün öğelerini kendi amacına göre tanımlar.

Arayüzler soyut yapılar olduğu için, doğrudan kullanılamazlar. Onları bir `nesne` (object) için yaptığımız gibi `referans` (işaret) edebiliriz. Bu referanslar ya `null` ya da arayüzü oluşturan `nesneyi` işaret ederler.

C# dilinde, en tepedeki `Object` sınıfı dışında, bir sınıfın ancak bir tane *atası* (taban, base) olabilir. Ama sınıflar ve arayüzler istenildiği kadar oğul üretebilirler. Bu olgu, C++ dilinde var olan, ama java ve C# dilinde olmayan *çoklu kalıtımın* (multiple inheritance) işlevinin daha iyi yapılmasını sağlar.

Bunu şöyle bir örnekle açıklayabiliriz. *Kuşlar* ve *Uçaklar* uçarlar, ama ataları aynı değildir. Onları ayrı ayrı birer sınıf olarak tanımlayabiliriz. Programcı, onları, diyelim ki, `Uçanlar` adlı bir üst sınıftan üretmek zorunda değildir. Ama her ikisi *Uçanlar* adlı bir arayüz oluşturabilir.

Arayüz bildirimi için başlık sözdizimi şöyledir:

```
public inteface IAd
{
}
```

Arayüz daima `public` erişim belirtecini gendeğer (default) olarak alır; öteki erişim nitelemelerini alamaz. O nedenle `public` nitelemesinin konulması ya da konulmaması sonucu değiştirmez. Başlıkta `interface` anahtar sözcüğünden sonra arayüzün adı yazılır. Arayüz adının ilk harfi daima `(I)` olmak zorundadır. Arayüzün gövdesine arayüz öğelerinin yalnızca imzaları (signature) yazılır; bu öğelerin tanımı o arayüzü kullanan (implement eden) sınıf içinde yapılır. Dolayısıyla, bir arayüzü kullanan sınıf, arayüzdeki bütün öğeleri tanımlamış olmalıdır. Aşağıdaki, yalnızca bir metot başlığı içeren basit bir arayüzdür.

```
interface Iaaa
{
    void bbb();
}
```

Bu arayüzün adı `Iaaa` 'dır. Arayüzün bir tek öğesi vardır: `bbb()` metodu. `void bbb()` ifadesi `bbb()` metodunun imzasıdır (signature). Anımsayacaksınız, metodun imzasında metodun adı, değer kümesinin veri tipi ve varsa parametreleri belirtilir. *Arayüz gövdesine* yalnızca metodun imzası yazılır; tanımı arayüzde değil, onu yaratacak sınıfta yapılır:

```
 class ccc : Iaaa
{
    // Arayüzdeki öğe(ler)in açık tanımı
    void Iaaa.bbb()
    {
        // Metot tanımı
    }
}
```

Tabii, arayüzü çağıracak `Main()` metodu herhangi bir sınıfta tanımlanabilir:

```
class Uygulama
{
    static void Main()
    {
        // Arayüzün bir nesnesini yaratır
        Iaaa obj = new ccc();
```

```
        // Arayüzün metodunu çağır
        obj.bbb();
    }
}
```

Bütün bunları bir araya getirirsek, şu program ortaya çıkar:

Arayüz02.cs

```
using System;

interface Iaaa
{
    void bbb();
}

class ccc : Iaaa
{
    // Arayüzdeki öğe(ler)in açık tanımı
    void Iaaa.bbb()
    {
        Console.WriteLine("bbb() metodu arayüzden çağrıldı.");
    }
}

class Uygulama
{
    static void Main()
    {
        // Arayüzün bir nesnesini yaratır
        Iaaa obj = new ccc();

        // Arayüzün metodunu çağır
        obj.bbb();
    }
}
```

Çıktı

```
bbb() metodu arayüzden çağrıldı.
```

Bu basit programı satır satır çözümlersek, arayüzlerin yapılışını ve kullanımını öğrenmiş olacağız. Bu örnek için yaptığımız gözlemleri, bütün arayüzlere genelleştirebiliriz.

interface Iaaa

Arayüzün imzasıdır. Bildirim *interface* anahtar sözcüğü ile başlar, adıyla devam eder.

Arayüze erişim belirteci konmadı. Arayüzün gendeğer (default) erişim belirteci *public* 'dir; niteleme konulmadığında otomatik olarak *public* nitelemesini alır. Başka niteleme yazılırsa derleme hatası doğar.

Arayüzün adının ilk harfi `'I'` dır. Bu harf, derleyiciye sözkonusu ad'ın bir arayüz adı olduğunu bildirir. `'I'` harfi yazılmadığında derleme hatası doğmaz, ama bunu yazmak iyi bir alışkanlıktır. Iaaa gibi bir ad görünce, onun bir arayüz olduğunu hemen anlarız.

void bbb();

Arayüzün öğe(ler)i yalnızca o öğe(ler)in imzasını taşır. Öğelerin tanımları arayüzü kullanan (implement eden) sınıfta yapılır.

```
class ccc : Iaaa
```

ccc sınıfı Iaaa arayüzünü kullanan (implement eden) sınıftır; görünüşte *ccc* sınıfı *Iaaa* arayüzünden türemiştir. Ama *Iaaa* arayüzünün öğelerinin tanımlandığı yer *ccc* sınıfıdır.

```
Iaaa obj = new ccc();
```

Herhangi bir sınıftaki `Main()` metodu içinde arayüzün bir nesnesi (object) yaratılabilir. Bu referans ile arayüze ait öğe(ler) çağrılabilir.

ccc sınıfının bir nesnesi içinden *Iaaa* arayüzünün öğelerine erişilemez.

Şimdi şu denemeyi yapalım. ccc sınıfı içindeki

```
void Iaaa.bbb()
```

imzasını

```
void bbb()
```

biçiminde yazalım ve derlemeyi deneyelim. Derleyici şu hata iletisini gönderir:

```
Error 1 'ccc' does not implement interface member 'Iaaa.bbb()'. 'ccc.bbb()' cannot implement an interface member because it is not public...
```

Şimdi de aynı metodu public nitelemesini verelim:

```
public void bbb()
```

Bu kez programın hatasız derlendiğini göreceğiz.

Aşağıdaki örnek bir noktanın koordinatlarını gösterecek iki özgen (property) içermektedir. Programı satır satır çözümleyiniz.

Arayüz03.cs

```
using System;

interface Inokta // Interface oluşturma
{
    // Özgenlerin imzaları
    int x
    {
        get;
        set;
    }

    int y
    {
        get;
        set;
    }
```

```
}
```

Arayüzü implement eden sınıf:

```
class Nokta : INokta
{
    // Alanlar (Fields):
    private int _x;
    private int _y;

    // Kurucular (Constructor):
    public Nokta(int x, int y)
    {
        _x = x;
        _y = y;
    }

    // Özgenlerin oluşumu (Property implementation):
    public int x
    {
        get
        {
            return _x;
        }

        set
        {
            _x = value;
        }
    }

    public int y
    {
        get
        {
            return _y;
        }
        set
        {
            _y = value;
        }
    }
}

class Uygulama
{
    static void NoktaYaz(INokta p)
    {
        Console.WriteLine("x={0}, y={1}", p.x, p.y);
    }

    static void Main()
    {
```

```
        Nokta p = new Nokta(2, 3);
        Console.Write("Noktanın koordinatları: ");
        NoktaYaz(p);
    }
}
```

Çıktı

```
Noktanın koordinatları: x=2, y=3
```

Arayüzler için şu kurallar geçerlidir:

Bir arayüzün dendeğer erişim belirteci public 'dir.
Bir arayüz'ün bütün öğelerinin erişim belirteci public 'dir.
Bir arayüz, bir yapı(struct)'dan veya bir sınıf(class)'tan kalıtımla türetilemez.
Bir arayüz, başka bir arayüzden ya da başka arayüzlerden kalıtımsal olarak türetilebilir.
Arayüz öğeleri static nitelemesini alamaz.
Arayüzü kullanan bir sınıf, arayüzün bütün öğelerinin açık tanımlarını vermek zorundadır.
Arayüzü kullanan sınıfta, arayüzün bütün öğeleri public erişim belirtecini alır.
Arayüz, nesnelere erişimi standart biçime sokar; nesnenin iç yapısını bilmeden nesneyi kullanma olanağı yaratır.

Arayüzden Arayüz Türetme

Bazan arayüzden arayüz türetmemiz gerekebilir. Bu iş, sınıftan sınıf türetmek kadar kolaydır. Aşağıdaki program bu işi yapmaktadır.

```
using System;

interface IAnaInterface
{
    void f();
}

interface IOğulInterface : IAnaInterface
{
    void g();
}

class aaaSınıf : IOğulInterface
{
    static void Main()
    {
        aaaSınıf iImp = new aaaSınıf();
        iImp.g();
        iImp.f();
    }

    public void g()
    {
        Console.WriteLine("g() metodu çağrıldı.");
    }

    public void f()
```

```
    {
        Console.WriteLine("f() metodu çağrıldı.");
    }
}
```

Okuma Parçası

Kullanıcı Arayüzü

Kullanılan herhangi bir alet için kullanıcıya bazı kolaylıklar sağlanır. Örneğin, bir fotoğraf makinasının bir televizyonun, bir otomobilin, bir uçağın, bir geminin yönetilmesi sırasında, kullanıcı, kullandığı aletin içindeki onlarca ya da on binlerce parçanın nasıl çalıştığını, o parçalar arasındaki etkileşimin nasıl yapıldığını bilmek zorunda değildir. O, yalnızca belirli işleri yaptıran düğmelere basmak, kolları çevirmek, pedallara basmak gibi kolay işleri yapar. Bir otomobil sürücüsünü alalım. Sürücü kontak anahtarını çevirince motor çalışır. Motorun çalışması için yakıt deposunun, yakıt pompasının, karbüratörün, silindirlerin, ateşleme sisteminin nasıl çalıştıklarını, kendi aralarında nasıl bir uyum sağladıklarını bilmeyebilir. Vites değiştirince aktarma organlarındaki dişlilerin nasıl yer değiştirdiğini, gaz pedalına basınca otomobilin neden hızlandığını, frene basınca aracın neden durduğunu bilmeyebilir. Bir düğmeye basmak, bir kolu çevirmek, arka arkaya bir çok işin sırayla ve önceden tasarlanmış bir düzen içinde yapılmasına neden olur.

Aletle insan arasında bu iletişimi kuran şey bir kullanıcı arayüzüdür. Bazı arayüzler gerçek anlamda kullanıcı dostudur, kolay öğrenilir, kolay kullanılırlar. Bazıları daha zor olabilir. Örneğin, ilk otomobillerde kontak anahtarı ve pistonlara ilk hareketi veren elektrikli mekanizma yoktu. Motoru çalıştırmak için, kullanıcının, kol kuvvetiyle pistonları harekete geçirmesi gerekiyordu. Benzer olarak bazı otomobillerde camlar bir düğmeyle açılıp kapanır, bazılarında ise mekanik bir kolla yapılır. Camı açan düğme ve kol birer kullanıcı arayüzüdür. Arayüzlerin bazısı çok kolay kullanılır, bazısı daha zor kullanılır.

Bilgisayar programları için de benzer şeyi düşünebiliriz. Yeni kuşak programlarda, grafiksel kullanıcı arayüzü (Graphical User Interface) denilen bir görüntü ekrana gelir. Örneğin, *Windows İşletim Sistemi* açılınca önümüze gelen ekran bir arayüz görüntüsüdür. *MSWord* programını ya da *Excel* programını açtığımızda önümüze gelen görüntüler arayüzlerin görüntüleridir. Ekrandaki göstergeden seçilen bir düğmeye basınca, kullanıcının asla görmediği program parçaları belli bir sıra ve uyum içinde geri planda çalışır ve kullanıcının istediğini yapar. Bu programlar için, geride çalışan modüllere hiç dokunmadan, farklı arayüzler hazırlanabilir. Mekanik aletlerde olduğu gibi, bilgisayar programlarında da bazı arayüzler gerçek anlamda kullanıcı dostudurlar, kolay öğrenilirler, kolay kullanılırlar. Bazıları ise daha az kullanıcı dostu olabilirler. Kullanıcı arayüzünün iyi ya da kötü oluşu, programı oluşturan modüllerle ilgili değildir. Aynı program için, çok iyi bir arayüz hazırlanabileceği gibi, kötü bir arayüz de hazırlanabilir.

Bilgisayar programları için çok çeşitli arayüzler vardır. Bunlar kendi aralarında Grafiksel Kullanıcı Arayüzleri (GUI- Graphical User Interfaces), Web Tabanlı Arayüzler (WUI- Web User Interfaces), Komut Arayüzleri (Command Line Interfaces), Dokunmatik Arayüzler (Touch Interfaces), Ses Arayüzleri (Voice Interfaces), Uygulama Programı Arayüzü (API- Application Program Interface), vb. sınıflara ayrılır.

Bölüm 20

Koleksiyonlar

Koleksiyon Sınıfları

Klâsik programlama dillerinde `array` çok önemli bir veri tipidir. Çok sayıda değişkeni kolayca tanımlar ve o değişkenlere istemli (random) erişim sağlar. Ancak *array* tipinin iki önemli handikapı vardır:

1. Array 'in öğeleri aynı veri tipinden olmalıdır,
2. Array 'in boyutu (öğe sayısı) önceden belli edilmelidir.

Oysa, programlama işinde, çoğunlukla aynı veri tipinden olmayan topluluklarla karşılaşırız. C# bu tür toplulukları ele alabilmek için, array yapısından çok daha genel olan koleksiyon (collection) veri tipini getirmiştir. Koleksiyon veri tpi, array veri tipinin çok kullanışlı bazı özelliklerini herhangi bir nesneler topluluğuna taşıma olanağı sağlamıştır.

Koleksiyon sınıfları nesnelerden oluşan topluluklardır. C#, koleksiyonları oluşturmak, koleksiyona yeni öğe katmak, koleksiyondan öğe atmak, koleksiyonun öğelerini sıralamak, numaralamak, koleksiyon içinde öğe aramak vb işleri yapmamızı sağlayan sınıflar, metotlar ve arayüzlerden oluşan çok geniş bir kütüphaneye sahiptir. Ayrıca, kullanıcı kendi koleksiyonunu oluşturabilir, onlarla istediği işi yapmayı sağlayacak metotları ve arayüzleri oluşturabilir.

C# dilinde koleksiyonlar `System.Collections` aduzayı (namespace) içinde birer veri tipidir. Klasik dillerdeki array yapısının çok daha gelişmiş biçimleridir. Bu bölümde C# dilinin koleksiyonlarından bazılarını oluşturacağız. Burada yapacaklarımız, başka koleksiyonların kullanımı için de yeterli olacaktır.

Ön-tanımlı koleksiyonlar

`System.Collections` aduzayı (namespace) içinde hemen kullanılmaya hazır koleksiyonlar vardır. Onlardan bazılarını ele alacağız. Bunların ilki `ArrayList` koleksiyonudur.

ArrayList Sınıfı

`ArrayList` sınıfı objelerden oluşan array yaratır. Array 'e bir obje eklemek için `Add()` metodu kullanılır.

Koleksiyon01.cs

```
using System;
using System.Collections;

namespace Koleksiyonlar
{
    class Dizi
    {
        static void Main(string[] args)
        {
            ArrayList birDizi = new ArrayList();
            birDizi.Add(12);
            birDizi.Add(3);
            birDizi.Add(8);
            birDizi.Add(7);
            birDizi.Add(15);

            foreach (int n in birDizi)
                Console.WriteLine(n.ToString());
        }
    }
}
```

ArrayList 'in öğelerini sıralamak (sort) için `Sort()` metodunu kullanırız. Aşağıda önce sıralanmamış liste, sonra sıralanmış liste yazılmaktadır.

Koleksiyon02.cs

```
using System;
using System.Collections;

namespace Koleksiyonlar
{
    class Dizi
    {
        static void Main(string[] args)
        {
            ArrayList birDizi = new ArrayList();
            birDizi.Add("Zonguldak");
            birDizi.Add("Urfa");
```

```
            birDizi.Add("Adana");
            birDizi.Add("Bursa");
            birDizi.Add("İzmir");
            Console.WriteLine("Sıralanmamış Liste");
            foreach (string s in birDizi)
                Console.WriteLine(s.ToString());
            Console.WriteLine();
            Console.WriteLine("Sıralanmış Liste");
           birDizi.Sort();
            foreach (string s in birDizi)
                Console.WriteLine(s.ToString());
        }
    }
}
```

Aşağıdaki program `ArrayList` 'i yaratmak için `ArrayYap()` metodunu ve yaratılan array'in öğelerini listeleyen `ArrayYaz()` metotlarını tanımlamıştır. Kodları dikkatle inceleyiniz.

Koleksiyon03.cs

```
using System;
using System.Collections;

namespace Koleksiyonlar
{
    class Diziler
    {
        public static void DiziYap(ArrayList arr)
        {
            for (int k = 1; k <= 10; k++)
              arr.Add(k);
        }

        public static void DiziYaz(ArrayList arr)
        {
            foreach (int n in arr)
                Console.WriteLine(n.ToString());
        }
    }

    class Uygulama
    {
        static void Main(string[] args)
        {
            ArrayList birDizi = new ArrayList();
            Diziler.DiziYap(birDizi);
            Diziler.DiziYaz(birDizi);
        }
    }
}
```

Yaratılan bir `ArrayList`' ten istenen öğeler atılabilir. Bunun için `Remove()` metodu kullanılır. Aşağıdaki program, yukarıda oluşan array'in bir öğesini attıktan sonra klan öğeleri tekrar listelemektedir.

Koleksiyon04.cs

```
using System;
using System.Collections;

namespace Koleksiyonlar
{
    class Diziler
    {
        public static void DiziYap(ArrayList arr)
        {
            for (int k = 1; k <= 10; k++)
              arr.Add(k);
        }

        public static void DiziYaz(ArrayList arr)
        {
            foreach (int n in arr)
                Console.WriteLine(n.ToString());
        }
    }

    class Uygulama
    {
        static void Main(string[] args)
        {
            ArrayList birDizi = new ArrayList();
            Diziler.DiziYap(birDizi);
            Diziler.DiziYaz(birDizi);
            birDizi.Remove(8);
            Diziler.DiziYaz(birDizi);
        }
    }
}
```

Alıştırma

Yukarıdaki programlarda `static` nitelemesi alan `ArrayYap()` ve `ArrayYaz()` metotlarını dinamik kılarak; yani static nitelemelerini kaldırarak kodları yeniden yazıp çalışır hale getiriniz.

C# dilinin bütün tipleri (sınıfları) `Object` tipinden türetilmiştir; yani bütün sınıfların atası `Object` 'dir. Dolayısıyla, koleksiyonların nesnelerden (object) oluşuyor olması doğaldır. Bazı koleksiyonlar aynı tipten objeleri içerir. Bazıları farklı tipten objeleri içerebilir. Örneğin, `array` sınıfına ait bir nesne aynı tipten öğeleri içerebilir. Ama `ArrayList` sınıfına ait nesneler farklı tipten nesneleri içerebilmektedir. Aşağıdaki program farklı tiplerden oluşan bir `ArrayList` oluşturmaktadır.

Koleksiyon05.cs

```
using System;
using System.Collections;
using System.Collections.Generic;


namespace Collections
{
    class Koleksiyon
    {
        static void Main()
        {
```

```
            int i = 10;
            double d = 17.3;
            ArrayList arrayList = new ArrayList();
            arrayList.Add("Başkent");
           arrayList.Add(i);
           arrayList.Add(d);
            for (int index = 0; index < arrayList.Count; index++)
                Console.WriteLine(arrayList[index]);
        }
    }
}
```

ArrayList 'in başlangıç kapasitesi 16 dır, ama 17-inci öğe geldiğinde kapasite 1 artar ve her yeni öğe gelişte bu olgu tekrarlanır. Ancak, her yeni öğenin gelişinde ona bellekte bir yer ayırma ve geleni oraya yerleştirme performansı düşürecektir. O nedenle, istenirse, başlangıçta ArrayList 'in kapasitesi, capacity özgeni kullanılarak belirlenebilir. Başka bir seçenek olarak, ArrayList sınıfının aşkın bir kurucusu kullanılabilir. Aşağıdaki program, bunun nasıl yapılacağını göstermektedir.

Koleksiyon06.cs

```
using System;
using System.Collections;
using System.Collections.Generic;


namespace Collections
{
    class Koleksiyon
    {
         static void Main()
        {
            int i = 10;
            double d = 17.3;
            ArrayList arrayList = new ArrayList();
           arrayList.Capacity = 2;
            arrayList.Add("Başkent");
           arrayList.Add(i);
           arrayList.Add(d);
            for (int index = 0; index < arrayList.Count; index++)
                Console.WriteLine(arrayList[index]);
        }
    }
}
```

StringCollection Sınıfı

`StringCollection` sınıfı `IList` arayüzünü oldurur ve stringlerden oluşan `ArrayList`'e benzer. Aşağıdaki program `StringCollection` sınıfının nasıl kullanılacağını göstermektedir.

Koleksiyon07.cs

```
using System;
using System.Collections;
using System.Collections.Specialized;

class Koleksiyonlar
```

```
{
  static void Main()
  {
    StringCollection stringList = new StringCollection();
    stringList.Add("Manisa");
    stringList.Add("Konya");
    stringList.Add("Kayseri");
    stringList.Add("Van");

    foreach (string str in stringList)
    {
      Console.WriteLine(str);
    }

  }
}
```

StringDictionary Sınıfı

`StringDictionary` sınıfı, anahtarları string olan bir Hashtable 'dir. Hashtable genel olarak her tipten anahtar kabul eder. Dolayısıyla, `StringDictionary`, anahtarları `string` sınıfına kısıtlı bir Hashtable 'dir. Aşağıdaki örnek `StringDictionary` sınıfının nasıl kullanıldığını göstermektedir.

Koleksiyon08.cs

```
using System;
using System.Collections;
using System.Collections.Specialized;

class Test
{
    static void Main()
    {
        StringDictionary stringList = new StringDictionary();
        stringList.Add("A", "Manisa");
        stringList.Add("B", "Konya");
        stringList.Add("C", "Kayseri");
        stringList.Add("D", "Van");

        foreach (string str in stringList.Values)
        {
            Console.WriteLine(str);
        }

    }
}
```

Stack Sınıfı

`Stack` sınıfı programlamada LIFO (Last-in-First-out , son giren ilk çıkar) diye bilinen önemli bir yapıdır. Bunu üst üste yığılan bir kitap koleksiyonu gibi düşününüz. Yığındakilere zarar vermeden istenen kitaba ulaşmak için, en üstteki kitap alınır ve bu işlem istenen kitaba erişinceye kadar devam eder. Stack 'a bir öğe eklemek için `Push()` metodu, stack 'ın üstünden bir öğe çekmek için `Pop()` metodu kullanılır. Aşağıdaki program `Stack` sınıfının nasıl kullanıldığını göstermektedir.

Koleksiyon09.cs

```
using System;
using System.Collections;

class Test
{
    static void Main()
    {
        Stack stackObject = new Stack();
        stackObject.Push("Sinop");
        stackObject.Push("Diyarbakır");
        stackObject.Push("Edirne");
        stackObject.Push("Muş");
        stackObject.Push("Aydın");
        stackObject.Push("Trabzon");
        while (stackObject.Count > 0)
            Console.WriteLine(stackObject.Pop());
    }
}
```

Queue Sınıfı

`Queue` sınıfı programlamada FIFO (First-in-First-out , ilk giren ilk çıkar) diye bilinen önemli bir yapıdır. Bu tam anlamıyla gündelik hayatta yaşadığımız kuyruk oluşturmayı andırır. Banka veya alış-veriş merkezlerinin vezne kuyruğunun oluşması gibidir. Kuyruğa ilk giren, veznedeki işini biirip ilk çıkan olacaktır. Arkasındakiler de aynı sıraya uyacaklardır. `Queue` (kuyruk) yapısı `Stack` yapısının yaptığının tersini yapar. `Queue` 'nun sonuna bir öğe eklemek için `Enqueue()` metodu, önünden bir öğe çekmek için `Dequeue()` metodu kullanılır. Aşağıdaki program `Queue` sınıfının nasıl kullanıldığını göstermektedir.

Koleksiyon10.cs

```
using System;
using System.Collections;

class Test
{
    static void Main()
    {
        Queue queueObject = new Queue();
        queueObject.Enqueue("Lale");
        queueObject.Enqueue("Gül");
        queueObject.Enqueue("Menekşe");
        queueObject.Enqueue("Sümbül");
        queueObject.Enqueue("Çiğdem");
        queueObject.Enqueue("Yasemin");
        while (queueObject.Count > 0)
            Console.WriteLine(queueObject.Dequeue());
    }
}
```

BitArray Sınıfı

`BitArray` sınıfı bit'lerden oluşan array yaratır. Aşağıda gösterildiği gibi, bit arraylerine *true* veya *false* değeri atanabilir.

```
BitArray bitArray = new BitArray(5,false);
```

ya da

```
BitArray bitArray = new BitArray(5,true);
```

Yukarıdaki sınıflarda olduğu gibi `BitArray` sınıfının da, öğe sayısını bildiren `Count` özgeni vardır. `BitArray` sınıfı öğeleri arasında aşağıdaki mantıksal işlemleri yapar.

```
And
Or
Not
Xor
```

Hashtable Sınıfı

`Hashtable` sınıfı, `Object` tiplerin hızla depolanması ve depodan hızla çekilmesi için iyi yöntemleri olan bir yapıdır. Anahtarlara dayalı arama yapar. Anahtarlar belli tiplere hasredilmiş hash kodlardan ibarettir. `GetHashCode()` metodu yaratılan bir nesnenin hash kodunu verir. Aşağıdaki program parçası `Hashtable` sınıfının nasıl kullanıldığını göstermektedir.

Koleksiyon11.cs

```
using System;
using System.Collections;

class Test
{
    static void Main()
    {
        Hashtable hashTable = new Hashtable();
        hashTable.Add(1, "Gökova");
        hashTable.Add(2, "Belek");
        hashTable.Add(3, "Çamdibi");
        hashTable.Add(4, "Marmaris");
        Console.WriteLine("Anahtarlar:--");
        foreach (int k in hashTable.Keys)
        {
            Console.WriteLine(k);
        }

        Console.WriteLine("Aramak için anahtarı giriniz :");
        int n = int.Parse(Console.ReadLine());
        Console.WriteLine(hashTable[n].ToString());
    }
}
```

`Hashtable` objelerini listelemek için, yukarıdaki listeleme yöntemi yerine, `IdictionaryEnumerator` 'i kullanarak aşağıda gösterildiği gibi de yapabiliriz.

Koleksiyon12.cs

```
using System;
using System.Collections;

class Test
{
    static void Main()
    {
        Hashtable hashTable = new Hashtable();
        hashTable.Add(1, "Matematik");
        hashTable.Add(2, "Fizik");
        hashTable.Add(3, "Kimya");
        hashTable.Add(4, "Biyoloji");
        hashTable.Add(5, "Bilgisayar");
        hashTable.Add(6, "Jeoloji");
        Console.WriteLine("Anahtarlar:--");
        IDictionaryEnumerator en = hashTable.GetEnumerator();
        string str = String.Empty;

        while (en.MoveNext())
        {
            str = en.Value.ToString();
             Console.WriteLine(str);
        }
    }
}
```

Hashtable sınıfından bir öğe atmak için Remove() metodu kullanılır. Örneğin,

```
hashTable.Remove(4) ;
```

deyimi, aşağıdaki listeden "Biyoloji" yi atacaktır.

SortedList Sınıfı

`SortedList` sınıfı `System.Object` tiplerini anahtar-değer çiftine göre yerleştirir; ayrıca sıralama yapar. Aşağıdaki program bunu gösteriyor.

Koleksiyon13.cs

```
using System;
using System.Collections;
using System.Collections.Specialized;

class Test
{
    static void Main()
    {
        SortedList sortedList = new SortedList();
        sortedList.Add(1, "Matematik");
        sortedList.Add(2, "Fizik");
        sortedList.Add(3, "Kimya");
        sortedList.Add(4, "Biyoloji");
        sortedList.Add(5, "Bilgisayar");
        sortedList.Add(6, "Jeoloji");

        IDictionaryEnumerator en = sortedList.GetEnumerator();
```

```
        Console.WriteLine("Listeyi giriliş sırasıyla yazar:");
        foreach (string str in sortedList.Values)
        {
            Console.WriteLine(str);
        }

    }
}
```

Çıktı

Listeyi giriliş sırasıyla yazar:

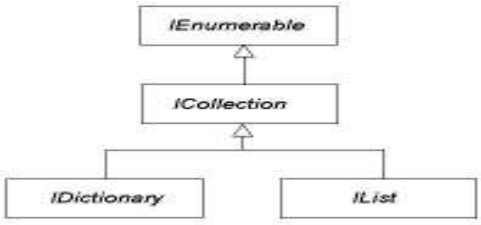
```
Matematik
Fizik
Kimya
Biyoloji
Bilgisayar
Jeoloji
```

Aynı listeyi `IdictionaryEnumerator` 'i kullanarak da yazdırabiliriz.

Koleksiyon14.cs

```
using System;
using System.Collections;
using System.Collections.Specialized;

class Test
{
    static void Main()
    {
        SortedList sortedList = new SortedList();
        sortedList.Add(1, "Matematik");
        sortedList.Add(2, "Fizik");
        sortedList.Add(3, "Kimya");
        sortedList.Add(4, "Biyoloji");
        sortedList.Add(5, "Bilgisayar");
        sortedList.Add(6, "Jeoloji");

        IDictionaryEnumerator en = sortedList.GetEnumerator();
        string str = String.Empty;
        while (en.MoveNext())
        {
            str = en.Value.ToString();
            Console.WriteLine(str);
        }
    }
}
```

Bütün koleksiyonlar *IEnumerable* arayüzünü oldururlar. *IEnumerable* arayüzü bütün koleksiyonların atasıdır. *ICollection* arayüzü `IEnumerable` arayüzünden türemiştir; yani oğuldur. Onun da iki tane oğulu vardır: `Idictionary` ve `IList`.



`IList` arayüzü değerler (value) koleksiyonudur. Onun oldurduğu koleksiyonlar şunlardır:

System.Array
System.Collections.ArrayList
System.Collections.Specialized.StringCollection

`IDictionary` arayüzü (Anahtar, Değer) [(Key, Value)] çiftlerinden oluşan koleksiyonlardır. Bunun oldurduğu koleksiyonların listesi şudur:

System.Collections.Hashtable
System.Collections.Specialized.ListDictionary
System.Collections.SortedList
System.Collections.Specialized.HybridDictionary

`ICollection` arayüzünden türetilen başka koleksiyonlar:

System.Collections.BitArray
System.Collections.Stack
System.Collections.Queue
System.Collections.Specialized.NameValueCollection

Bunların dışında olan koleksiyonlar da vardır. Örneğin, System.Collections.Specialized.StringDictionary, System.Collections.Specialized.BitVector32. Bunların ayrıntıları için *msdn* web sitesine bakınız.

Koleksiyon15.cs

```
using System;
using System.Collections;
using System.Collections.Generic;


namespace Collections
{
    class Koleksiyon
    {
        static void Main()
        {
            String[] names =new String[2] {"Joydip","Jini"};
for(IEnumerator e =
```

```
names.GetEnumerator();e.MoveNext();Response.Write(e.Current));

String[] adlar = new String[2] {"Joydip","Jini"};
foreach(string str in adlar)
Response.Write(str);
        }
    }
}
```

Alıştırma

Aşağıdaki program bir liste yaratmaktadır. Çıktısı ile karşılaştırarak programı çözümleyiniz.

Koleksiyon16.cs

```
using System;
public class ListeYap
{
    static int[] list = new int[5];
    static int sayaç = 0;
    static void add(int k)
    {
        if (sayaç == list.Length)
        {
            int[] newlist = new int[(int)(sayaç +1)];
            for (int i = 0; i < sayaç; i++)
              newlist[i] = list[i];
           list = newlist;
        }
        list[sayaç++] = k;
    }
    public static void Main(string[] args)
    {
        for (int i = 1; i <= 20; i++)
           add(i);
        Console.WriteLine("length = " + sayaç);
        Console.Write("kents = ");
        for (int i = 0; i < sayaç; i++)
            Console.Write(list[i] + " ");
        Console.WriteLine();
    }
}
```

Çıktı

```
length = 20
elements = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Bölüm 21

Delegeler (delegates)

`Delegeler` metot işaret eden referanslardır. Bunun ne anlama geldiğini örneklerle açıklayacağız.

Bir metodu çağırmak için, hiç bir aracıya (delege) gerekseme duymayan iyi bir yöntemimiz vardır. Şimdiye kadar yapageldiğimiz yöntemi aşağıdaki örnek üzerinde anımsayalım.

Aracısız Metot Çağırma

Delege01.cs

```
using System;

namespace Teknikler
{
    class Vergiler
    {
        public double BrütOku()
        {
            Console.WriteLine("Brüt Geliri Giriniz : ");
            return Double.Parse(Console.ReadLine());
        }

        public double GelirVergisi(double d)
        {
            if (d > 20000)
                return d * 0.40;
            else
                return d * 0.25;
```

```
        }
    }
    class Uygulama
    {
        public static void Main(string[] args)
        {
            Vergiler v = new Vergiler();
            Console.WriteLine(v.GelirVergisi(v.BrütOku()));
        }
    }
}
```

Bu programı çözümlemek kolaydır. `Vergiler` sınıfının iki metodu var. `BrütOku()` adlı metot, kullanıcının gireceği brüt geliri okuyor. `GelirVergisi` adlı metot ise, Brüt gelir 20000 den fazla ise %40 , değilse %25 gelir vergisi hesaplıyor. *Uygulama* sınıfındaki *Main()* metodu *Vergiler* sınıfının `v` adlı bir nesnesini yaratıyor. Sonra v nesnesi içinden `GelirVergisi()` metodunu çağırıyor. O da BrütOku() metodunu çağırıyor. Son satırda iç-içe yuvalanmış metotları çözümlemekte bir zorluk yoktur.

Şimdi bu metotları delegelerle çağırmayı öğreneceğiz. İlkönce delege bildirimini yapmalıyız.

Basit Bir Delege Bildirimi

Delege bildirimi metot imzası gibidir. Sözdizimi şöyledir:

```
delegate değer-bölgesi ad([parametreler]);
```

Açıklama

delegate	delege bildirimi için anahtar sözcük
değer-bölgesi	delegenin işaret edeceği metodun değer-bölgesi
ad	delegenin adı
parametreler	işaret edeceği metodun bağlı olduğu parametreler (varsa)

Örnekler:

```
public delegate void BasitDelege () ;
```

Bu bildirimde verilen *BasitDelege* adlı delege parametresiz ve *void* değer alan metotları işaret eder.

```
public delegate int DüğmeyeTıkla (object obj1, object obj2) ;
```

Bu bildirimde verilen `DüğmeyeTıkla` adlı delege object tipinden iki parametreye bağlı olan ve int değer alan metotları işaret eder.

Delege, hangi sınıftan hangi metodu çağıracağımızı bilmeden, çağrılacak metodun neye benzediğini belirlememizi sağlar. Biz, yalnızca, işaret etmesini istediğimiz metodun imzasını belirtiyoruz.

Delege bildirimi ve kullanımı üç aşamalıdır:

delegenin Bildirimi

Delegenin yaratılması

Delegenin çağrılması

Delege İle static Metot Çağırma

Aşağıdaki program delege ile metot çağırmanın çok basit bir örneğidir.

Delege02.cs

```
using System;

namespace Delegeler
{
    public delegate void BasitDelege();

    class Uygulama
    {
        public static void Topla()
        {
            Console.WriteLine("12 + 8 =" + (12+8));
        }

        public static void Main()
        {
            BasitDelege obj = new BasitDelege(Topla);
            obj();
        }
    }
}
```

Üçüncü satırda delege bildirimi yapılmıştır:

```
      public delegate void BasitDelege();
```

Uygulama sınıfındaki `Topla()` metodu 12+8 işlemini yapıp konsola yazmaktadır.

Main() metodunun ilk satırında delegeye ait obj adlı bir nesne yaratılmıştır. Bu nesne *Topla()* metodunu işaret etmektedir.

```
BasitDelege obj = new BasitDelege(Topla);
```

Son satırda ise yaratılan nesne çağrılmıştır. obj nesnesinin çağrılması demek, işaret ettiği metodun çağrılması demektir. *Obj* nesnesinin İşaret ettiği metot *Topla()* metodudur. Sonuç olarak, son satırdaki kod, *Topla()* metodunu çalıştırmaktadır.

Programda delegenin bildiriminin *Topla()* metodunun imzasına benzediğine dikkat ediniz.

```
public delegate void BasitDelege();
public static void Topla()
```

Her ikisinin değer kümesi `void` 'dir. Her ikisinin parametreleri yoktur. Bu demektir ki, delegenin değer kümesi ve parametreleri çağıracağı fonksiyonunkilerle aynı olmalıdır. Gerçekten, birisinin değer kümesi belirtecinde `void` yerine `int, string` gibi başka bir veri tipi koyarsanız, derleyici,

```
Error    1 'void Delegeler.Uygulama.Topla()' has the wrong return type...
```

hata mesajını verir. Bu mesaj, delege ile işaret edeceği `Topla()` metodunun değer kümelerinin uyuşmadığını söylemektedir. Benzer olarak, eğer delege bildiriminde `BasitDelege(int n)` biçiminde bir parametre koyarsanız, derleyici,

```
Error    1 No overload for 'Topla' matches delegate ...
```

hata mesajını verir. Bu mesaj, delege ile işaret edeceği `Topla()` metodunun parametrelerinin

uyuşmadığını söylemektedir.

Bu örnekte, `Topla()` metodu `static` nitelemeli olduğu için, onu, ait olduğu sınıfın bir nesnesini yaratmadan çağırabiliyoruz. Eğer `static` nitelemesini kaldırırsak, derleyici,

```
Error   1   An object reference is required for the non-static field, method, or property...
```

hata mesajını verir. Bu mesaj, static olmayan *Topla()* metodunun bellekteki adresini işaret eden bir referans gerektiğini söylemektedir. Başka bir deyişle, onu ancak ait olduğu `Uygulama` sınıfına ait bir nesneden çağırabiliriz. Aşağıdaki örnek bunu yapmaktadır.

Delege İle Dinamik Metot (member method) Çağırma

Delege03.cs

```
using System;

namespace Delegeler
{
    public delegate void BasitDelege();

    class Uygulama
    {
        public  void Topla()
        {
            Console.WriteLine("12 + 8 =" + (12+8));
        }

        public static void Main()
        {
            Uygulama u = new Uygulama();
            BasitDelege obj = new BasitDelege(u.Topla);
            obj();
        }
    }
}
```

İki programı satır satır karşılaştırarak, static nitelemesi kaldırılınca Topla() metodunun çağrılabilmesi için nasıl nesne yaratıldığını görünüz.

Delege İle Parametreli Metot Çağırma

Aşağıdaki programda parametreli delege ile parametreli metot çağrılmaktadır. Burada delege ile `Topla()` metodunun değer kümelerinin ve parametrelerinin aynı olduğuna dikkat ediniz. Son satırdaki `obj(12,8)` çağrısında kullanılan 12 ve 8 parametreleri `Topla()` metoduna geçmektedir.

Delege04.cs

```
using System;

namespace Delegeler
{
    public delegate void BasitDelege(int a, int b);

    class Uygulama
    {
```

```
        public  void Topla(int x, int y)
        {
            Console.WriteLine("{0} + {1} = {2} " , x ,y , x+y);
        }

        public static void Main()
        {
            Uygulama u = new Uygulama();
            BasitDelege obj = new BasitDelege(u.Topla);
            obj(12,8);
        }
    }
}
```

Aşağıdaki program, `void` yerine `int` koyarak yukarıdaki programı değiştirmiştir. Bunun için `Topla()` metoduna bir `return` değerini koymak yetmiştir.

Delege05.cs

```
using System;

namespace Delegeler
{
    public delegate int BasitDelege(int a, int b);

    class Uygulama
    {
        public  int Topla(int x, int y)
        {
            Console.WriteLine("{0} + {1} = {2} " , x ,y , x+y);
            return x + y;
        }

        public static void Main()
        {
            Uygulama u = new Uygulama();
            BasitDelege obj = new BasitDelege(u.Topla);
            obj(12,8);
        }
    }
}
```

Aşağıdaki programdaki `Delege01` adlı delege, `Vergiler` sınıfında tanımlanan `BrütOku` metodunu çağırmaktadır. Çağrılan metot, kullanıcının girdiği `double` değerini almaktadır. Aynı sınıfta tanımlı olan `GelirVergisi()` metodu `Main()` metodu tarafından ne doğrudan ne de delege ile çağrılmaktadır.

Delege06.cs

```
using System;

namespace Teknikler
{
    public delegate double Delege01();
    class Vergiler
    {
```

```
        public double BrütOku()
        {
            Console.WriteLine("Brüt Geliri Giriniz : ");
            return Double.Parse(Console.ReadLine());
        }

        public double GelirVergisi(double d)
        {
            if (d > 20000)
                return d * 0.40;
            else
                return d * 0.25;
        }
    }
    class Uygulama
    {
        public static void Main(string[] args)
        {
            Vergiler v = new Vergiler();
            Delege01 obj01 = new Delege01(v.BrütOku);
            Console.WriteLine(obj01());
        }
    }
}
```

Delege01 için yapılan aşamalar şunlardır:

Üçüncü satırda delege bildirimi yapılmıştır:

```
public delegate double Delege01();
```

Main() metodunun birinci satırı Vergiler sınıfının v adlı bir nesnesini yaratmıştır:

```
Vergiler v = new Vergiler();
```

Main() metodunun ikinci satırı Delege01 delegesinin obj01 adlı bir nesnesini yaratmıştır. Bu nesne v.BrütOku() metodunu işaret etmektedir:

```
Delege01 obj01 = new Delege01(v.BrütOku);
```

Main() metodunun son satırında obj01 nesnesinin işaret ettiği v.BrütOku metodu çağrılmıştır.

```
Console.WriteLine(obj01());
```

Şimdi, Vergiler sınıfındaki iki metodu çağıran birer delege oluşturalım. Delegelerin birisi parametresiz, diğeri double tipinden bir parametrelidir.

Delege07.cs

```
using System;

namespace Teknikler
{
    public delegate double Delege01();
    public delegate double Delege02(double dd);

    class Vergiler
    {
```

```
        public double BrütOku()
        {
            Console.WriteLine("Brüt Geliri Giriniz : ");
            return Double.Parse(Console.ReadLine());
        }

        public double GelirVergisi(double d)
        {
            if (d > 20000)
                return d * 0.40;
            else
                return d * 0.25;
        }
    }
    class Uygulama
    {
        public static void Main(string[] args)
        {
            Vergiler v = new Vergiler();
            Delege01 obj01 = new Delege01(v.BrütOku);
            Delege02 obj02 = new Delege02(v.GelirVergisi);
            Console.WriteLine(obj02(obj01()));
        }
    }
}
```

Bu programı çözümleyelim.

Üçüncü ve dördüncü satırda iki tane delege bildirimi yapılmıştır:

```
    public delegate double Delege01();
    public delegate double Delege02(double dd);
```

Vergiler sınıfında `BrütOku()` ve `GelirVergisi()` adlı iki metot tanımlanmıştır.

Main() metodunun birinci satırı *Vergiler* sınıfının `v` adlı bir nesnesini yaratmıştır:

```
Vergiler v = new Vergiler();
```

Main() metodunun ikinci satırı Delege01 delegesinin obj01 adlı bir nesnesini yaratmıştır. Bu nesne v.BrütOku() metodunu işaret etmektedir:

```
Delege01 obj01 = new Delege01(v.BrütOku);
```

Main() metodunun üçüncü satırı Delege02 delegesinin obj02 adlı bir nesnesini yaratmıştır. Bu nesne v.GelirVergisi metodunu işaret etmektedir:

```
Delege02 obj2 = new Delege02(v.GelirVergisi);
```

Main() metodunun son satırında obj02 nesnesi obj01 nesnesini parametre olarak almıştır; dolayısıyla v.BrütOku metodu çağrılmıştır. Onun getirdiği brüt miktar obj02 nesnesinin işaret ettiği GelirVergisi metoduna parametre olarak geçmiş ve istenen vergi hesabı yapılmıştır.

```
Console.WriteLine(obj02(obj01()));
```

Alıştırma

Aşağıdaki çok basit bir delege kullanımı örneğidir. Programı satır satır çözümleyiniz.

Delege08.cs

```
using System;
namespace Delegeler
{
    /*
     * Bu delege int tipi iki parametresi olan ve int
     * değer alan her metodu işaret edebilir
*/
    public delegate int Delegem(int x, int y);

    //Bu sınıf Delegem tarafından işaret edilebilecek metotlar içeriyor

    public class Hesap
    {
        public static int Topla(int a, int b)
        {
            return a + b;
        }
        public static int Çarpım(int a, int b)
        {
            return a * b;
        }
    }

    class Uygulama
    {
        static void Main(string[] args)
        {
            //Hesap.Topla() metodunu işaret eden Delegem nesnesi yarat
            Delegem delege1 = new Delegem(Hesap.Topla);

            //Delegate kullanarak  Topla() metodunu çağır
            int toplam = delege1(7, 8);
            Console.WriteLine("7 + 8 = {0}\n", toplam);

            // Hesap.Çarpım() metodunu işaret eden Delegem nesnesi yarat
            Delegem delege2 = new Delegem(Hesap.Çarpım);

            //Delegate kullanarak Çarpım() metodunu çağır
            int çarpım = delege2(7, 8);
            Console.WriteLine("7 X 8 = {0}", çarpım);
        }
    }
}
```

Çıktı

```
7 + 8 = 15

7 X 8 = 56
```

Bölüm 22

Boxing – Unboxing

Kutulama (boxing) Nedir?
Kutulama Nasıl Yapılır?
Kutu Nasıl Açılır?

Kutulama Nedir?

C# dilinde veri tiplerinin *değer-tipleri (value types)* ve *referans-tipleri (reference type)* olmak üzere ikiye ayrıldığını biliyoruz.

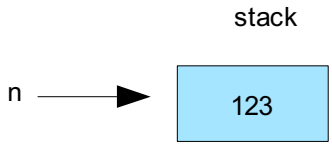
Değer tiplerinin referans tipine dönüşmesine *kutulama (boxing)*, referans tiplerin değer tipine dönüşmesine ise *kutu-açma (unboxing)* denir. Bu dönüşümler, daha önce veri tipleri arasında yaptığımız istemsiz (implicit) ve istemli (explicit) dönüşümlerin özel bir durumudur; yani tiplerden birinin nesne (object) olduğu durumdur.

Değer tiplerinin referans tipine dönüştürülmesiyle, nesne üzerinde yapılan bazı işlemlerin dönüşen değer tipine de uygulanabilmesi olanağı doğar. O nedenle, kutulama ve kutu-açma eylemleri C# dilinde önemlidirler.

C# dili, değer tiplerini *stack* içinde, referans tiplerini *heap* içinde tutar. Örneğin,

```
int n = 123;
```

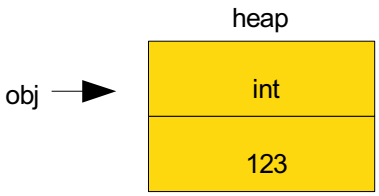
deyimi, n değişkeni için *stack* içinde bir adres ayırır, 123 değeri o adrese yerleşir, `n` ise o adresi işaret eden referanstır.



Öte yandan

```
object obj = n;  // istemsiz kutulama (implicit boxing)
```

deyimi istemsiz (implicit) bir dönüşümdür. Deyim `object` tipinden bir nesne yaratır ve `123` sayısını o nesne içine yerleştirir. `obj` o nesneyi işaret eden referanstır. `obj` referans tipi olduğu için, onun işaret ettiği adres *heap* içindedir. Bu eylem 123 sayısını bir kutuya koymaya benzer. "`Boxing`" kutulama, paketleme anlamındadır.



Kutulama (boxing) ve kutu-açma (unboxing) eylemleri değer tiplerinin nesne imiş gibi işlem görmesini sağlar. Bir değer tipini kutulayınca (*boxing)* edince `object` sınıfının bir nesnesi içine gömülür (paketlenir). Böyle olunca, söz konusu değer tipi *heap* içinde depo edilmiş olur. Daha önemlisi, o değer artık bir nesne gibi kullanılabilir.

Kutu-açma *(unboxing)* ise tersini yapar. Nesne içine gömülen değeri nesnenin dışına alır:

```
n = (int)obj;  // kutu-açma (unboxing)
```

Bu istemli (explicit) bir dönüşümdür; daha önce yaptığımız 'casting' eylemidir. `obj` referansının işaret ettiği nesne içindeki değeri `int` tipine dönüştürmektedir.

Kutulama (boxing) ve kutu-açma (*unboxing)* dönüşümleri performans azaltıcı eylemlerdir. Çünkü, *boxing* yaparken heap içinde bir nesne yaratılıyor, yani ana bellekte bir yer ayrılıyor ve *stack* 'taki değer oraya kopyalanıyor.

Kutu-açma (unboxing) eylemi biraz daha az olmakla birlikte, nesneden değere dönüşüm (casting) için bir zaman harcar ve performansı düşürür. C# 2.0 sürümünden sonra bu sorunu aşan yöntemler getirilmiştir. Özellikle `System.Collections.Generic` adyeri (namespace), *boxing* ve *unboxing* yapmaya gerek bırakmayan çok sayıda sınıf, metot ve arayüz tanımlamıştır. Performansın önemli olduğu zamanlarda, o adyerine başvurulması gerekir. C# dilinin ilk sürümlerinde önemli bir araç olarak kullanılan *boxing* ve *unboxing* eylemlerini, performansın önemli olmadığı durumlarda başvurulan kolay yöntemler olduğu için bilmekte yarar vardır. Bu eylemleri birkaç örnekle göstereceğiz.

Değer Tipleri (Value Types)

Değer tipleri ilkel tiplerdir, doğrudan FCL'e gönderilir. Örneğin *Int32* tipi *System.Int32* ye gider, *double* tipi `System.Double` ' a gider. Bütün değer tipleri *stack* içinde depolanır. Bütün değer tipleri

`System.ValueType` sınıfından türerler. `System.ValueType` 'dan tütetilen bütün *yapılar* ve *enumerated* (numaralanmış) tipler *stack* içinde yaratılırlar.

Referans Tipleri (Reference Types)

Bütün referans tiplere *heap* içinde yer ayrılır. Bütün sınıflar referans tipidir. `new` operatörü referansların işaret edeceği adresleri *heap* içinde ayırır.

Boxing01.cs

```
class TestBoxing
{
    static void Main()
    {
        int n = 123;
        object obj = n;  // istemsiz kutulama (implicit boxing)

        n = 456;         // n 'nin değerini değiştir

        System.Console.WriteLine("Değer-tipi değeri = {0}", n);
        System.Console.WriteLine("nesne tipi değeri = {0}", obj);
    }
}
```

Çıktı

```
Değer-tipi değeri = 456

nesne tipi değeri = 123
```

Bu programı çözümleyelim.

```
int n = 123 ;
```

deyimi n değişkenine *stack* içinde bir adres ayırıyor ve o adrese 123 sayısını yerleştiriyor.

```
object obj = n;
```

deyimi istemsiz kutulama yaparak *heap* içinde *obj* referansı ile işaret edilen bir nesne yaratıyor ve oraya 123 değerini yerleştiriyor. Bu aşamada hem *stack* 'ta hem *heap* 'te 123 tutuluyor.

```
n = 456;
```

atama deyimi, `n` 'nin *stack* içindeki adresine 456 değerini yazıyor. Bu atamadan sonra, *stack* 'ta 123 silinip, yerine 456 konulmuş oluyor.

Son iki satır *stack* 'taki ve *heap* 'teki verileri konsola yazıyor. Bunların farklı oluşu, kutulama eyleminin 123 sayısına ana bellekte farklı bir yer ayırdığını gösterir. Eğer farklı adres ayırmasaydı, son iki satır aynı çıktıyı verirdi.

Veri tipi değişse de sonuç aynıdır. Programdaki `int` yerine `string` konulursa, aşağıdaki gibi benzer sonuç elde edilir.

Boxing02.cs

```
class TestBoxing
{
    static void Main()
    {
        string s = "Ankara" ;
        object obj = s;  // istemsiz kutulama (implicit boxing)

        s = "İstanbul";  // s 'nin değerini değiştir

        System.Console.WriteLine("Değer-tipi değeri (stack 'taki değer) =
{0}", s);
        System.Console.WriteLine("Nesne tipi değeri (heap 'teki  değer) =
{0}", obj);
    }
}
```

Çıktı

```
Değer-tipi değeri (stack 'taki değer) = İstanbul
Nesne tipi değeri (heap 'teki  değer) = Ankara
```

Örnekler

Bütün değer-tipleri System.ValueType sınıfından türerler, dedik. Başka bir deyişle, System.ValueType sınıfı bütün değer-tiplerinin atasıdır, onların taban sınıfıdır. Bunun ne anlama geldiğini örnekler üzerinde açıklayalım. Yukarıdaki programda int n = 123 yerine

```
System.ValueType n = 123 ;
```

koyalım. Programı derleyip koşturursak, sonucun aynı olduğunu görebiliriz. Peki ama böyle olması ne anlama gelir? Yukarıdaki deyim

```
System.Int32  n = 123 ;
```

deyimine denk midir? Denk olup olmadığını, bu deyimi programda yazarak deneyebiliriz. Gerçekten int n = 123 deyimi yerine System.ValueType n = 123 deyimini yazdığımızda da aynı sonucu elde edeceğimizi deneyerek görebiliriz. Buradan şu sonuç çıkıyor:

```
int n = 123 ;
System.ValueType n = 123 ;
System.Int32 n = 123 ;
```

deyimleri denktir. `System.ValueType`, temsil ettiği veri tipiyle ilgili işlemlere girebilir. Örneğin,

```
int n = 5 ; n++ ;
```

deyimi

```
System.ValueType n = 5 ; n++ ;
```

deyimine denktir.

Şimdi aklımıza gelmesi gereken bir soru var. `System.ValueType n = 123` deyimi neden `System.Int32 n = 123` deyimine denktir? Başka bir deyişle, neden `System.Int16 n = 123` ya da `System.Int64 n = 123` deyimine denk değildir? Bu sorunun yanıtı şudur: `int n = 123` deyimi yazıldığında, derleyici `n` sayısını default olarak `Int32` 'ye gönderir. Her veri tipi için

derleyicinin belirlediği bir *başlangıç-tipi* vardır. Tamsayılar için bu tip `Int32` dir. Gerçekten böyle olduğunu görmek isterseniz, `GetType()` metodunu kullanabilirsiniz. Aşağıdaki program bazı veri tiplerinin başlangıç tiplerini yazmaktadır.

Boxing03.cs

```
using System;
class TestBoxing
{
    static void Main()
    {
        System.ValueType n = 123 ;
        Console.WriteLine(n.GetType()); // System.Int32
        //-------------------------------------------------------
        System.ValueType r = 23.45;
        Console.WriteLine(r.GetType()); // System.Double
        //-------------------------------------------------------
        System.ValueType x = 23.45F;
        Console.WriteLine(x.GetType()); // System.Single
        //-------------------------------------------------------
        System.ValueType y = 2U;
        Console.WriteLine(y.GetType()); // System.UInt32
        //-------------------------------------------------------
        System.ValueType b = false;
        Console.WriteLine(b.GetType()); // System.Boolean
        //-------------------------------------------------------
    }
}
```

System.ValueType 'dan object tipine istemsiz (implicit) dönüşümü C# kendiliğinden yapar. Bu demektir ki, istemsiz kutulama (implicit boxing) eylemi kendiliğinden olur. Ama istenirse, istemli (explicit) dönüşüm; yani istemli kutulama (explicit boxing) de yapılabilir. Aşağıdaki program bunu yapmaktadır.

Boxing04.cs

```
using System;

class TestBoxing
{
    static void Main()
    {
        Int32 x = 10;
        object obj1 = x;  // istemsiz kutulama (implicit boxing)
        Console.WriteLine(" obj1 nesnesi = {0}", obj1); // Çıktı : 10
        //-------------------------------------------------------------
        Int32 y = 10;
        object obj2 = (object)y; // istemli kutulama (Explicit Boxing)
        Console.WriteLine(" obj2 nesnesi  = {0}", obj2); // Çıktı : 10
    }
}
```

Kutudan başka tip çıkar mı?

Kutulanan bir değer, başka bir tipe dönüşmüş olarak açılabilir mi? Örneğin, *Int32* olarak kutulanmış bir veriyi kutudan *Int64* olarak geri alabilir miyiz? Bu sorunun yanıtı "hayır" dır. Veri hangi tip olarak kutulanmışsa, kutu açılınca aynı tip olarak çıkar.

Aşağıdaki program bunu östermektedir. Program derlenir, ama koşmaz, *RunTime* hatası doğar. Çünkü `Int32` tipi kutulanıyor (boxing), ama kutuyu açarken (unboxing) içeriğinin `Int64` tipinden olması isteniyor. Buna izin verilmez. Kutuya hangi tip konulduysa, kutu açılınca aynı tip çıkacaktır.

Boxing05.cs

```
using System;

class TestBoxing
{
    static void Main()
    {
        Int32 x = 123;     // Int32 tip değişken bildirimi
        Int64 y ;          // Int64 tip değişken bildirimi
        object obj = x;    // istemsiz Kutulama (Implicit Boxing)
        y = (Int64)obj;    // hata !
        Console.WriteLine("y={0}", y);
    }
}
```

Uyarı mesajı:

```
Unhandled Exception: System.InvalidCastException: Specified cast is not valid.
```

Tabii, şimdiye dek öğrendiklerimizle bu sorunun üstesinden nasıl gelebileceğimizi biliyoruz. Önce kutu-açma (unboxing) eylemi verinin asıl tipi olan *Int32* tipine yapılır. Sonra kutudan çıkan değere istemli (explicit) dönüşüm (casting) uygulayabiliriz.

Boxing06.cs

```
using System;

class TestBoxing
{
    static void Main()
    {
        Int32 x = 123;     // Int32 tip değişken bildirimi
        Int64 y ;          // Int64 tip değişken bildirimi
        object obj = x;    // istemsiz Kutulama (Implicit Boxing)
        y = (Int64)(Int32)obj; // double tipine istemli dönüşüm (casting)
        Console.WriteLine("y={0}", y);
    }
}
```

Kutuyu istemli dönüşümle açabilir miyiz?

Şimdi şunu deneyelim. Kutularken istemsiz ve istemli dönüşümler yaptık. Acaba kutuyu açarken de istersek istemsiz istersek istemli dönüşüm yapabilir miyiz? Zaten yukarıdaki örneklerde hep istemli dönüşümle kutuyu açtık; yani sorunun birisinin yanıtını biliyoruz: Kutu açarken istemli dönüşüm yapılabiliyor. Öyleyse, şimdi kutunun *istemsiz (implicit)* dönüşümle açılıp açılmayacağını deneyebiliriz. Aşağıdaki programı derlemeyi deneyelim.

Boxing07.cs

```
using System;

class TestBoxing
{
    static void Main()
    {
        Int32 x = 123;    // Int32 tip değişken bildirimi
        object obj = x;   // istemli Kutulama (Implicit Boxing)
        x = obj;          // istemli kutu-açma (implicit unboxing)
        Console.WriteLine("y={0}", x);
    }
}
```

Derleyici şu hata iletisini gönderecektir:

```
Error    1    Cannot implicitly convert type 'object' to 'int'. An explicit conversion exists (are you missing a cast?)...
```

Derleyicimiz, *nesne* tipinin *int* tipine istemsiz dönüşemeyeceğini, istemli dönüşüm gerektiğini söylüyor. *int* tipi yerine *string, bool* vb tipleri koyarsanız, gene benzer hata iletisini alırsınız. Demek ki, nesne tipinden herhangi bir `SystemValueType` tipine istemsiz dönüşüm yapılamaz.

Aşağıdaki program generic ve non-generic tiplerin hızlarını karşılaştırmaktadır.

Boxing08.cs

```
using System;
using System.Collections;
using System.Collections.Generic;

class Deneme
{
    static void Main(string[] args)
    {
        Deneme denek = new Deneme();
        denek.GenericsTest();
        denek.ArrayListTest();
    }

    public void GenericsTest()
    {
        int total = 0;
        List<int> li = new List<int>();
        for (int i = 0; i < 20; i++)
            li.Add(i);
        for (int i = 0; i < 10000000; i++)
            foreach (int el in li)
                total += el;
```

```
        Console.WriteLine(total);
    }

    public void ArrayListTest()
    {
        int total = 0;
        ArrayList al = new ArrayList();
        for (int i = 0; i < 20; i++)
             al.Add(i);
        for (int i = 0; i < 1000000; i++)
            foreach (int el in al)
                 total += el;
        Console.WriteLine(total);
    }
}
```

Alıştırmalar

Aşağıdaki programları koşturmadan, çıktılarını tahmin ediniz.

Boxing09.cs

```
using System;
class Test
{
    static void Main()
    {
        Console.WriteLine(7.ToString());
    }
}
```

Boxing10.cs

```
using System;
class Test
{
    static void Main()
    {
        int i = 1;
        object o = i;         // boxing
        Console.WriteLine(o);

        int j = (int)o;       // unboxing
        Console.WriteLine(j);

    }
}
```

Bölüm 23

Sıralama ve Arama

Sıralama ve Arama Kavramı

Sıralama ve arama (sorting/searching)) bilgisayar uygulamalarında çok önem taşır. Klâsik programlama dillerinde sıralama ve arama yapan kodları programcı kendisi yazardı. O nedenle, Bilgisayar uygulamalarında *Sıralama ve Arama Algoritmaları* denilen ayrı bir konu gelişmiştir. Yapılan işe göre en uygun algoritmayı seçmek programcının işiydi.

C# dili, programcının omuzundan bu yükü büyük ölçüde almıştır. Yalnız arrayleri değil, koleksiyonları sıralamak için kütüphanesine etkin yöntemler koymuştur. Özellikle metinlerin sıralanması alfabelere ve kültürlere bağlı olduğu için, `System.Globalization` aduzayı (namespace) sıralama işinde de devreye girer.

Aşağıda Türkçe alfabeye göre sıralama yapan bir program göreceksiniz.

Sıralama01.cs

```
using System;
using System.Globalization;
using System.Threading;
using System.Collections;
namespace Yöresellik
{
    class Sıralama01
    {
```

```
        static void Main(string[] args)
        {
            string[] kentler = {"Ünye", "ıhlara", "İzmir", "ğğğ" , "ĞĞĞ",
                        "Zonguldak", "Çanakkale", "Uşak",
                        "Kayseri", "Malatya", "çandır", "İstanbul",
                        "Ören", "Şanlıurfa", "Adana", "Afyon"};

            // arrayin sırasını değiştirmeden
            Console.WriteLine("\nSırasız...");
            adYaz(kentler);
            Console.WriteLine();

            // Türk alfabesine göre sıralanıyor
            Thread.CurrentThread.CurrentCulture =
               new CultureInfo("tr-TR");

            Array.Sort(kentler);
            Console.WriteLine("\nTürk Alfabesine göre Sıralı...");
            adYaz(kentler);

            // invariant kültüre göre
            Array.Sort(kentler, Comparer.DefaultInvariant);
            Console.WriteLine("\n DefaultInvariant kültüre göre
sıralama...");
            adYaz(kentler);
        }

        static void adYaz(IEnumerable e)
        {
            foreach (string s in e)
                Console.Write(s + " - ");
            Console.WriteLine();
        }
    }
}
```

Çıktı

```
Sırasız...
Ünye - ıhlara - İzmir - ğğğ - ĞĞĞ - Zonguldak - Çanakkale - Uşak - Kayseri - Mal
atya - çandır - İstanbul - Ören - Şanlıurfa - Adana - Afyon -

Türk Alfabesine göre Sıralı...
Adana - Afyon - Çanakkale - çandır - ğğğ - ĞĞĞ - ıhlara - İstanbul - İzmir - Kay
seri - Malatya - Ören - Şanlıurfa - Uşak - Ünye - Zonguldak -

DefaultInvariant kültüre göre sıralama...
Adana - Afyon - Çanakkale - çandır - ğğğ - ĞĞĞ - ıhlara - İstanbul - İzmir - Kay
seri - Malatya - Ören - Şanlıurfa - Ünye - Uşak - Zonguldak -
```

Türkçe Windows kullanıldığı için, *Türkçe Alfabe* 'ye göre sıralama ile *DefaultInvariant kültüre* göre sıralama aynı sonucu verir. İşletim Sistemi değişirse, bu iki sıralama farklı olabilir.

Aşağıdaki program `ArrayList` ile bir array yaratıyor, arrayi sıralıyor ve istenen bir bileşen değerinin hangi indisli bileşende olduğunu buluyor.

Sıralama02.cs

```
using System;
using System.Collections;

public class SortSearchDemo
{
    public static void Main()
    {
        // Bir ArrayList kur
        ArrayList arr = new ArrayList();

        // ArrayList'e bileşen ekle
        arr.Add(17);
        arr.Add(84);
        arr.Add(13);
        arr.Add(-46);
        arr.Add(3);
        arr.Add(-16);
        arr.Add(-8);
        arr.Add(9);
        arr.Add(21);
        arr.Add(-33);
        arr.Add(68);

        Console.Write("İlk ArrayList: ");
        foreach (int i in arr)
            Console.Write(i + " ");
        Console.WriteLine("\n");

        // Sırala
        arr.Sort();


        Console.Write("Sıralama sonrası ArrayList: ");
        foreach (int i in arr)
            Console.Write(i + " ");
        Console.WriteLine("\n");

        Console.WriteLine("-33 'ün indeksi = " +
                          arr.BinarySearch(-33));
    }
}
```

Çıktı

```
İlk ArrayList: 17 84 13 -46 3 -16 -8 9 21 -33 68
Sıralama sonrası ArrayList :  -46 -33 -16 -8 3 9 13 17 21 68 84
-33 'ün indeksi = 1
```

Ters Sıralama

Bazı durumlarda arrayi ters (azalan) yönde sıralamamız gerekir. Bunu yapan etkin yöntemler geliştirilebilir; ama boyu çok uzun olmayan arraylerde, artan yönde sıraladıktan sonra `Reverse()` metodunu kullanmak basitlik sağlar. Ancak bu yöntem çok büyük arraylerde bellek kullanımı ve zaman açısından yeğlenmez.

Sıralama03.cs

```
using System;
using System.Collections;
```

```
using System.Collections.Generic;
using System.Text;

class Program
{
    static void Main(string[] args)
    {
        ArrayList futbolTakımı = new ArrayList();
        futbolTakımı.Add("Fenerbahçe");
        futbolTakımı.Add("Galatasaray");
        futbolTakımı.Add("Beşiktaş");
        futbolTakımı.Add("Trabzon");

        Console.Write("Sıralamadan önce : \n");

        foreach (string takım in futbolTakımı)
        {
            Console.Write("\n" + takım + "\n");
        }
        futbolTakımı.Sort();
        futbolTakımı.Reverse();
        Console.Write("\n Sıralamadan sonra : \n");
        foreach (string takım in futbolTakımı)
        {
            Console.Write("\n" + takım + "\n");
        }
    }
}
```

Çıktı

```
Sıralamadan önce :

Fenerbahçe
Galatasaray
Beşiktaş
Trabzon

 Sıralamadan sonra :
Trabzon
Galatasaray
Fenerbahçe
Beşiktaş
```

Aynı yöntemi sayısal diziler için de kullanabiliriz.

Sıralama04.cs

```
using System.Collections;
using System.Collections.Generic;
using System.Text;

class Program
{
    static void Main(string[] args)
    {
        int[] dizi = new int[] { 3, 1, 4, 5, 2 };
```

```
        Array.Sort(dizi);
        Array.Reverse(dizi);

        Console.Write("Sıralamadan önce : \n");

        for (int i=0 ; i<dizi.Length ; i++)
        {
            Console.WriteLine(dizi[i]);
        }
    }
}
```

`Icomparable` arayüzü kullanılarak, ters sıralamayı yapan bir program yazılabilir. Aşağıda bir örnek görülmektedir.

Sıralama05.cs

```
using System;
using System.Collections.Generic;

namespace CSharp411
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] array = new int[] { 3, 1, 4, 5, 2 };
            ArraySorter<int>.SortDescending(array);
            WriteArray(array);
            Console.ReadLine();
        }
        static private void WriteArray(int[] array)
        {
            foreach (int i in array)
            {
                Console.WriteLine(i);
            }
        }
    }
    static public class ArraySorter<T>
        where T : IComparable
    {
        static public void SortDescending(T[] array)
        {
            Array.Sort<T>(array, s_Comparer);
        }
        static private ReverseComparer s_Comparer = new
ReverseComparer();
        private class ReverseComparer : IComparer<T>
        {
            public int Compare(T object1, T object2)
            {
                return -((IComparable)object1).CompareTo(object2);
            }
        }
    }
}
```

Arama Teknikleri

Bilgisayar programlarında sıralama gibi önemli olan öteki konu arama eylemidir. *Arama Algoritmaları* (Search Algorithms) programlamanın önemli araştırma alanlarından birisi olagelmiştir. Bu konuda çok etkili teknikler geliştirilmiştir. Çok kullanılanlardan birisi *binary search*, ötekisi *linear search* (doğrusal arama) tekniğidir. Doğrusal aramada, aranan değer arrayin başından sonuna doğru her öğesi ile karşılaştırılır. Bulduğunda arama durur. Bulamazsa sonuna kadar devam eder. *Binary search* tekniğinde ise, yarılama denilen bir yöntem kullanılır. Aranan öğe, arrayin başlarında ise doğrusal arama daha çabuk bulur. Değilse, binary search daha çabuk bulur. Aranan öğenin nerede olduğu bilinmediğine göre binary search tekniği tercih edilir.

Doğrusal Arama

Aşağıdaki program, kullanıcıdan aldığı verilerle bir tamsayı array oluşturuyor, sonra kullanıcının istediği bir veriyi array içinde arıyor. Bulursa, onun arrayin kaçıncı öğesi olduğunu , bulamazsa o sayının array içinde olmadığını konsola yazıyor. Kodların açıklamaları kaynak programa yazıldığı için, çözümlemeyi vermeye gerek yoktur.

DoğrusalArama01.cs

```
using System;

class Arama
{
    public static void DoğrusalArama()
    {
        //tamsayılardan oluşan bir array kur
        int[] numArray = new int[100];
        //Girilen veri sayı değilse
        bool isNum = false;
        //array'in boyutu
        int sizeNum;
        //kullanıcıdan arrayin boyutunu sor
        Console.WriteLine("Arrayin boyutunu giriniz");
        //Girilen sayıyı oku
        string sizeString = Console.ReadLine();
        //TryParse kullanarak girilen verinin sayısal olduğunu denetle
        isNum = Int32.TryParse(sizeString, out sizeNum);
        // true/false
        if (isNum)
        {
            //Arrayin öğelerini gir
            Console.WriteLine("-------------------------");
            Console.WriteLine("\n Arrayin sayısal öğelerini gir  \n");
            // girilen sayıları bir döngü ile oku
            for (int i = 0; i < sizeNum; i++)
            {
                int tempNum;
                // her bileşeni ayrı ayrı Enter ile gir
                string elements = Console.ReadLine();
                //TryParse ile girilen verinin sayısal olduğunu denetle
                isNum = Int32.TryParse(elements, out tempNum);
                if (isNum)
                {
                    //girilen veri sayısal ise, arraye ata
                    numArray[i] = tempNum;
                }
                else
                {
                    Console.WriteLine("Sayısal olmayan veri girdiniz!");
```

```
                    break;
                }
            }
            //Kullanıcıdan arayacağı sayıyı iste
            Console.WriteLine("-------------------------");
            Console.WriteLine("Aranacak sayıyı giriniz \n");
            //gririlen veriyi oku
            string searchString = Console.ReadLine();
            //girilecek sayıyı tutacak değişken
            int searchNum;
            isNum = Int32.TryParse(searchString, out searchNum);
            //Girilen veri sayısal mı?
            if (isNum)
            {
                //Arrayi döngü ile tara
                for (int i = 0; i < sizeNum; i++)
                {
                    //Eğer aranan değer, array içinde varsa
                    //kullanıcıya hangi indis olduğunu bildir.
                    if (numArray[i] == searchNum)
                    {
                        Console.WriteLine("-------------------------");
                        Console.WriteLine("Aradığınız {0} sayısı dizinin
{1} -inci öğesidir \n", searchNum, i + 1);
                        return;
                    }
                }
                //Aranan sayı bulunmamışsa
                Console.WriteLine("Aranan {0} sayısı bulunamadı \n",
searchNum);
            }
            else
            {
                //Kullanıcı aranacak veriyi sayısal girmediyse
                Console.WriteLine("Aranmak için sayısal bir değer
girmediniz!");
            }
        }
        else
        {
            Console.WriteLine("Lütfen bir sayı giriniz!");
        }
    }
    static void Main(string[] args)
    {
        DoğrusalArama();
    }
}
```

Binary Arama

Aşağıdaki program, verilen bir sayının sıralı bir array içinde olup olmadığını binary search yöntemiyle aramaktadır.

BinaryArama01.cs

```
using System;
```

```
class ArrayBinarySearch
{
    public static void Main()
    {
        int[] arr = { 10, 20, 30, 40, 50, 1000 };
        Console.WriteLine("Arrayin öğeleri: ");
        for (int i = 0; i < arr.Length; i++)
        {
            Console.Write("arr[{0}] = {1, -5}", i, arr[i]);
        }
        Console.WriteLine();
        FindObject(arr, 20);
        FindObject(arr, 35);
        FindObject(arr, 10000);
    }

    public static void FindObject(Array ary, Object o)
    {
        int index = Array.BinarySearch(ary, 0, ary.Length, o);
        Console.WriteLine();
        if (index > 0)
        {
            Console.WriteLine("Object: {0} -nin indisi :  [{1}]", o,
index);
        }
        else if (index == ary.Length)
        {
            Console.WriteLine("Object: {0} bulunamadı. "
               + "Arama bitti.", o);
            Console.WriteLine();
        }
        else
        {
            Console.WriteLine("Object: {0} bulunamadı. "
               + "Aramanın eriştiği indis : [{1}].", o, ~index-1);
        }
    }
}
```

Çıktı

```
Arrayin öğeleri:
arr[0] = 10   arr[1] = 20   arr[2] = 30   arr[3] = 40   arr[4] = 50   arr[5] = 1000
Object: 20 -nin indisi :  [1
Object: 35 bulunamadı. Aramanın eriştiği indis : [2]
Object: 10000 bulunamadı. Aramanın eriştiği indis : [5].
```

Programı çözümleme:

`ArrayBinarySearch` adlı sınıf içinde aramayı yapmak üzere

```
public static void FindObject(Array ary, Object o)
```

metodu tanımlanıyor. Bu metot Array sınıfının

```
BinarySearch(ary, 0, ary.Length, o)
```

metodunu kullanıyor. Bu metodun aşkın sürümleri var. Program aranan öğe bulunursa, bulunduğu bileşenin indisini yazıyor, bulamazsa, eriştiği bileşenin indisini yazıyor. Array sıralı olduğu için bunu yapması kolaydır. Aranan öğeden büyük değerli bir bileşene erişirse, aranan öğenin array içinde olmayacağı açıktır.

Bölüm 24

Indeksci (indexer)

İndeksci Nedir?

İndeksci bir sınıfa ait nesneleri sanki bir arrayin bileşenleri imiş gibi indeksler. Sınıfa ait öğelerin ne olduğu indeksci için önemli değildir. O yalnızca yaratılan nesnelere damga (indis) koyar. Sınıfa ait `özgen` (property) tanımlamaya çok benzer. O nedenle, ona *akıllı array* takma adı verilmiştir. İndeksçiler *private, public, protected ya da internal* erişim belirteçleriyle nitelenebilirler. Aşağıdaki program nesneleri tamsayı indisler veren basit bir indeksçidir.

Indeksci01.cs

```
using System;

    class IndesksYap
    {
        private string[] metin = new string[5];
        public string this[int i]
        {
            get
            {
                return metin[i];
            }
            set
            {
                metin[i] = value;
```

```
            }
        }
    }

    class NesneYap
    {
        public static void Main()
        {
            IndesksYap obj = new IndesksYap();
            obj[0] = "Konya";
            obj[1] = "Elma";
            obj[2] = "Şiir";
            obj[3] = "Okul ";
            obj[4] = "C# ile Nesne Programlama";
            for (int j = 0; j < 5; j++)
            {
                Console.WriteLine("{0}", obj[j]);
            }
        }
    }
}
```

Çıktı

```
Konya
Elma
Şiir
Okul
C# ile Nesne Programlama
```

Bu programı çözümleyerek indekscilerin nasıl çalıştığını açıklayabiliriz. `İndeksYap` sınıfındaki

```
private string[] metin = new string[5];
```

deyimi string türünden, metin adlı, 5 bileşenli, `private` nitelemeli bir `array` bildirimidir.

```
        public string this[int i]
        {
            get
            {
                return metin[i];
            }
            set
            {
                metin[i] = value;
            }
        }
```

blokunda `this` pointeri yaratılan arrayi işaret eder. `get/set` erişimcileri (accessors), arrayin bileşenlerine değer atar ve atanan değeri okur.

`NesneYap` sınıfı içindeki `Main()` metodu

```
IndesksYap obj = new IndesksYap();
```

deyimiyle `IndeksYap` sınıfına ait obj adlı bir nesne yaratır. Bu nesnenin string tipinden, 5 bileşenli bir array olduğunu biliyoruz. `Main()` metodunun sonraki kodları arrayin bileşenlerine değerler atıyor sonra o değerleri *for döngüsü* ile konsola yazdırıyor.

Aşağıdaki örnek `double` tipinden sayıları bir `array` gibi kullanmamızı sağlar.

Indeksci02.cs

```
using System;

class IntIndexer
{
    private double[] kesir;

    public IntIndexer(int boyu)
    {
        kesir = new double[boyu];

        for (int i = 0; i < boyu; i++)
        {
            kesir[i] = 100.0;
        }
    }

    public double this[int ndx]
    {
        get
        {
            return kesir[ndx];
        }
        set
        {
            kesir[ndx] = value;
        }
    }

    static void Main(string[] args)
    {
        int boyu = 10;

        IntIndexer aaa = new IntIndexer(boyu);

        aaa[7] = 12.8;
        aaa[2] = 5.3;
        aaa[4] = 6.0;

        for (int i = 0; i < boyu; i++)
        {
            Console.WriteLine("aaa[{0}] = {1} ", i, aaa[i]);
        }
    }
}
```

Çıktı

```
aaa[0] = 100
aaa[1] = 100
aaa[2] = 5,3
aaa[3] = 100
```

```
aaa[4] = 6
aaa[5] = 100
aaa[6] = 100
aaa[7] = 12,8
aaa[8] = 100
aaa[9] = 100
```

Programı çözümleyelim. `IntIndexer` sınıfındaki

```
private double[] kesir;
```

deyimi double tipinden kesir adlı ve private nitelemeli bir array yaratır. private nitelemeli olduğu için bu arraye sınıf dışından erişilemez. O nedenle erişimcileri (accessors) kullanıyoruz.

```
public IntIndexer(int boyu)
{
    kesir = new double[boyu];

    for (int i = 0; i < boyu; i++)
    {
        kesir[i] = 100.0;
    }
}
```

bloku `IntIndexer` sınıfının 1 parametreli bir kurucusudur. `kesir` arrayinin bütün bileşenlerine 100.0 değerini atıyor. Bu değer atanmasa da olur. O zaman `double` tipi bileşenler *öndeğer* (default value) olarak 0.0 değerini alırlar.

```
public double this[int ndx]
{
    get
    {
        return kesir[ndx];
    }
    set
    {
        kesir[ndx] = value;
    }
}
```

blokunda `this` pointeri kesir arrayini gösteriyor. Bu pointer yardımıyla `get/set` erişimcileri (accessor methods) arrayin bileşenlerine değer atıyor ve atanan değeri okuyor.

`Main()` metodu, ilkönce arrayin uzunluğunu `boyu = 10` deyi ile belirliyor. Sonra

```
IntIndexer aaa = new IntIndexer(boyu);
```

deyimi ile IntIndexer sınıfına ait aaa nesnesini yaratıyor. aaa içindeki kurucu, 10 bileşenli kesir adlı arrayi oluşturuyor. set metodu bütün bileşenlere 100.0 değerini atamıştı.

```
aaa[7] = 12.8;
aaa[2] = 5.3;
aaa[4] = 6.0;
```

deyimleri 7, 2 ve 4 damgalı bileşenlerin değerlerini değiştiriyor, yeni değerler atıyor. Bu kodlar, arrayin bileşenlerine atanan değerlerin istendiğinde değiştirilebileceğini gösteriyor.

`Main()` metodunun içindeki son blok for döngüsüyle arrayin bileşenlerinin değerlerini konsola yazdırıyor.

Aşkın İndeksci

İndeksçiler birer metot olduğuna göre, aşkın indeksçi olması doğaldır. Aşağıdaki program bir aşkın indeksçi (overloaded indexer) tanımlıyor.

Indeksci03.cs

```
using System;

class AIndexer
{
    private string[] kent;
    private int arrBoy;

    public AIndexer(int boy)
    {
        arrBoy = boy;
        kent = new string[boy];

        for (int i = 0; i < boy; i++)
        {
            kent[i] = "***";
        }
    }

    public string this[int ndx]
    {
        get
        {
            return kent[ndx];
        }
        set
        {
            kent[ndx] = value;
        }
    }

    public string this[string veri]
    {
        get
        {
            int sayaç = 0;

            for (int i = 0; i < arrBoy; i++)
            {
                if (kent[i] == veri)
                {
                    sayaç++;
                }
            }
            return sayaç.ToString();
        }
        set
        {
            for (int i = 0; i < arrBoy; i++)
```

```
            {
                if (kent[i] == veri)
                {
                    kent[i] = value;
                }
            }
        }
    }

    static void Main(string[] args)
    {
        int boy = 10;
        AIndexer aaa = new AIndexer(boy);

        aaa[9] = "Rize";
        aaa[3] = "Trabzon";
        aaa[5] = "Sinop";
        for (int i = 0; i < boy; i++)
        {
            Console.WriteLine("aaa[{0}]  =  {1}", i, aaa[i]);
        }
    }
}
```

Çıktı

```
aaa[0] = ***
aaa[1] = ***
aaa[2] = ***
aaa[3] = Trabzon
aaa[4] = ***
aaa[5] = Sinop
aaa[6] = ***
aaa[7] = ***
aaa[8] = ***
aaa[9] = Rize
```

Bu programı çözümleyelim. Aindexer sınıfındaki

```
private string[] kent;
```

deyimi private nitelemeli, string tipinden kent adlı bir array bildirimidir.

```
private int arrBoy;
```

deyimi, kent arrayinin bileşen sayısını tutacak bir değişken bildirimidir.

```
    public AIndexer(int boy)
    {
        arrBoy = boy;
        kent = new string[boy];

        for (int i = 0; i < boy; i++)
        {
            kent[i] = "***";
        }
    }
```

bloku `AIndexer` sınıfının int tipinden 1 parametreli bir kurucusudur. String tipinden kent arrayini yaratır ve bileşenlerine "***" değerini atar.

```
    public string this[string veri]
    {
        get
        {
            int sayaç = 0;

            for (int i = 0; i < arrBoy; i++)
            {
                if (kent[i] == veri)
                {
                    sayaç++;
                }
            }
            return sayaç.ToString();
        }
        set
        {
            for (int i = 0; i < arrBoy; i++)
            {
                if (kent[i] == veri)
                {
                    kent[i] = value;
                }
            }
        }
    }
```

blokunda this pointeri arrayi işaret eder. Dolayısıyla, ilk satır [string veri] parametresi ilk kurucunun int tipi parametre tipinden farklıdır. O halde, bu bir aşkın kurucudur. Bu bloktaki get/set erişimcileri (accessors) `kent` arrayinin bileşenlerine for döngüleriyle değer atıyor ve o değerleri okuyor.

`Main()` metodu arrayin bileşen sayısını 10 olarak belirledikten sonra `Aindexer` sınıfının `aaa` adlı bir nesnesini yaratıyor. Sonra arrayin 9, 3, 5 damgalı bileşenlerine değer atıyor ve bütün bileşen değerlerini for döngüsü ile konsola yazdırıyor.

Alıştırma

Aşağıdaki programın deyimlerini çıktı ile karşılaştırarak çözümleyiniz.

Indeksci04.cs

```
using System;
//using System.Collections.Generic;
//using System.Text;

namespace Indexers
{
    class Ata
    {
        private string[] bölge = new string[5];

        public string this[int indexbölge]
```

```
        {
            get
            {
                return bölge[indexbölge];
            }

            set
            {
                bölge[indexbölge] = value;
            }
        }
    }

    /* Ata sınıfının nesnelerini bir array gibi kullanır */
    class Oğul
    {
        public static void Main()
        {
            Ata obj = new Ata();

            obj[0] = "Matematik";
            obj[1] = "Fizik";
            obj[2] = "Kimya";
            obj[3] = "Biyoloji";
            obj[4] = "İstatistik";

            Console.WriteLine("{0}\n,{1}\n,{2}\n,{3}\n,{4}\n", obj[0],
obj[1], obj[2], obj[3], obj[4]);
        }
    }
}
```

Çıktı

```
Matematik
,Fizik
,Kimya
,Biyoloji
,İstatistik
```

Bölüm 25

Numaratör (enumerator)

Numaratör Nedir?

Nesneleri numaralamak için kullanılır. C# var olan temel veri tiplerine ek olarak, programcı, gerekseme duyduğunda sınıf, struct, ve array veri tiplerini yaratabiliyordu. *Numaratör (enumarate)* de istediğinde programcının yarattığı bir veri tipidir. Tabii, bir veri tipi yaratırken, programcı belirli bir işi yapmak ister. Numaratör bir liste içindeki öğeleri numaralamaya yarar. Bunun nasıl olduğunu aşağıdaki örneklerle göreceğiz.

Enumerator01.cs

```
using System;

enum Aylar : byte
{
    Ocak, Şubat, Mart, Nisan, Mayıs, Haziran
}
class Enumerate01
{
    public static void Main()
    {
        byte a = (byte)Aylar.Ocak;
        byte b = (byte)Aylar.Şubat;
        byte c = (byte)Aylar.Mart;
        byte d = (byte)Aylar.Nisan;
        byte e = (byte)Aylar.Mayıs;
        byte f = (byte)Aylar.Haziran;
        Console.WriteLine("Ocak={0} , Nisan = {1}, Haziran={2}", a, d, f);
    }
 }
```

Çıktı

```
Ocak=0 , Nisan = 3, Haziran=5
```

Çözümleme:

Numaratör (enumerate) daima

```
enum Aylar : byte
```

deyimine benzer bir başlıkla başlar.

enum	numaratör için anahtar sözcüktür.
Aylar	numaralanacak öğelere verilen addır.
: byte	koleksiyonun numaralarının byte veri tipinden olacağını belirtir.

```
enum Aylar : byte
{
    Ocak, Şubat, Mart, Nisan, Mayıs, Haziran
}
```

bloku içindeki {Ocak, Şubat, Mart, Nisan, Mayıs, Haziran } kümesi, numaralanacak öğeleri belirten listedir. enum bu listenin öğelerini sırayla numaralıyor. Bu blok Aylar 'ı bir veri tipi yapar. Numaralama, aksi istenmedikçe öndeğer (default) olarak 0 dan başlar birer artarak Aylar'a ait öğelere sırayla birer numara verir. Verilen numaralar, öğe adlarının yerine geçer, bir tür takma ad olurlar; yani öğenin asıl adı ile numarası birbirlerini kesinlikle belirlerler: Mart'ın numarası 2 dir ve 2 numaralı öğe Mart'tır.

```
(byte)Aylar.Ocak;
```

deyimi Aylar veri tipinin Ocak adlı öğesine verilen numaradır. Bu numarayı byte tipinden bir değişkene atayabiliriz:

```
byte a = (byte)Aylar.Ocak;
```

Artık bu numarayı, istersek konsola yazdırabiliriz. Yukarıdaki Main() metodu bu işi yapmaktadır.

Numaratör için sözdizimi şöyledir:

```
<belirteç> enum <enum_adı>
{
  // Numaralanacak öğelerin listesi
}
```

Numaralama istenen bir sayıdan başlatılabilir.

```
using System;
enum Hafta: long
{
    pzt = 1, sal, çar, per, cm, cmt, paz
}
class Uygulama
{
    public static void Main()
    {
        byte a = (byte)Hafta.pzt;
        byte b = (byte)Hafta.sal;
        byte c = (byte)Hafta.çar;
```

```
        Console.WriteLine("Pazartesi = {0} , Salı = {1}, Çarşamba = {2}",
a, b, c);
    }
}
```

İstersek, numaralamayı kendimiz belirleyebiliriz. Yukarıdaki programı şöyle yazalım:

Enumerator03.cs

```
using System;
enum Meyve: long
{
    elma=3 , armut = 7, çilek = 17
}
class Uygulama
{
    public static void Main()
    {
        byte a = (byte)Meyve.elma;
        byte b = (byte)Meyve.armut;
        byte c = (byte)Meyve.çilek;

        Console.WriteLine("Armut = {0} , Elma = {1}, Çilek = {2}", a, b,
c);
    }
}
```

Çıktı

```
elma = 3 , armut = 7, çilek = 5
```

İstenirse bir numara birden çok öğeye de verilebilir:

Enumerator04.cs

```
using System;
enum Ağaç:int
{
    Çam = 3, KızılÇam = 3, Ladin =1,   Meşe = 7
}
class Uygulama
{
    public static void Main()
    {
        int x = (int)Ağaç.Çam;
        int y = (int)Ağaç.KızılÇam;
        int z = (int)Ağaç.Ladin;
        int w = (int)Ağaç.Meşe;

        Console.WriteLine("Çam = {0} , KızılÇam = {1}, Ladin = {2} , Meşe
= {3}", x, y, z, w);
    }
}
```

Numaratör (enum) vereceği numaralar için şu veri tiplerini kullanabilir: `byte, sbyte, short, ushort, int, uint, long, or ulong.`

Numaratörün taban öndeğer (default) tipi `int` veri tipindendir. Dolayısıyla, bildirim anında onu yazmayabiliriz:

Enumerator05.cs

```
using System;
enum Mevsim
{
    İlkBahar = 1, Yaz, SonBahar,   Kış
}
class Uygulama
{
    public static void Main()
    {
        int a = (int)Mevsim.İlkBahar;
        int b = (int)Mevsim.Yaz;
        int c = (int)Mevsim.SonBahar;
        int d = (int)Mevsim.Kış;

        Console.WriteLine("İlkbahar = {0} , Yaz = {1}, Sonbahar = {2} ,
Kış = {3}", a, b, c, d);
    }
}
```

enum tipler üzerinde kısıtlar:

1. enum blokunda metot tanımlanamaz
2. Arayüz kullanamzlar (implemet)
3. enum blıkunda özgen (property) ve indeksçi tanımlanamaz.

Bölüm 26

Çıktıyı Biçemli Yazdırma

Neden Biçemleme
Sayıların Biçemlenmesi
`Write()` ya da `WriteLine()` metotları ile çıktı elde etme
Sağa/sola Yanaşık Yazdırma
{} Yer Tutucu ile Biçemleme
Standart Sayısal Biçem Belirtgenleri
Simgesel biçimler (picture formats)
Üstel Notasyon
`ToString()` metodunu kullanmak
`NumberFormatInfo`

Neden Biçemleme

Bilgisayar çıktısının kolay okunur ve kolay anlaşılır biçeme (format) konulması uygulamada önem taşır. Özellikle, sayılar ve tarihler farklı ülke, farklı alfabe ve farklı kültürlerde farklı biçimlerde yazılır. Örneğin, biz kesirli sayıların kesir kısmını (ondalık kısım) `kesir ayıracı` dediğimiz virgül (,) ile ayırırız. Tam kısmı çok haneli olan sayıları kolay okumak için, sayının tam kısmının hanelerini sağdan sola doğru üçerli gruplara ayırırız. Bu işe sayıyı binliklerine ayırmak diyoruz. Örneğin,

```
123.456,789
```

sayısı, bizim için tamsayı kısmı yüz yirmi üç bin dörtyüz elli altı ve kesirli sayı (ondalık) kısmı binde yedi yüz seksen dokuz olan bir sayıdır. Ama aynı anlama gelmesi için bu sayı İngiltere'de veya ABD'de

```
123,456.789
```

biçiminde yazılmalıdır.

C# dili java dilinde olduğu gibi, karakterleri 16 bitlik `Unicode` sistemiyle yazar. Dolayısıyla, dünyadaki bütün dillere, alfabelere ve kültürlere hizmet edebilme yeteneğine sahip gelişkin bir dildir. Bu demektir ki, C# dili metinleri ve sayıları istediğimiz alfabede ve istediğimiz biçemde (format) yazabilir.

Bu kitap, konuları sistematik anlatmak yerine pedagojik anlatma yöntemini seçmiştir. O nedenle, çıktıların nasıl biçemlendiğinin sistematiğine çok girmeden, pratik uygulamalarla konuyu açıklayacağız.

Sayıların Biçemlenmesi

C# dilinde `tamsayı` veri tiplerini 8, `kesirli sayı` tiplerini de 3 ayrı gruba ayırmıştık. `.NET` onların herbirisini `System` aduzayında (namespace) içinde ayrı birer sınıf olarak tanımlar. Dolayısıyla her bir sınıfın kendisine özgü metotları vardır.

Çıktıyı biçimlendirmek için üç yöntem kullanırız.

1. `System.Console` sınıfı içindeki `Write()` ya da `WriteLine()` metotlarını kullanmak.
2. Her sınıfta var olan `ToString()` metodunu kullanmak.
3. `String.Format()` metodunu kullanmak.

Özellikle, hemen her sınıfta ortak ad taşıyan `ToString()` metodunun sayısal veri tiplerinin (sınıflarının) hepsinde olduğunu unutmayalım. Bu metodu kullanarak, sayıların yerel kültürlere uyan çıktılarını elde edebiliriz. Bunu yaparken sayısal verileri string'e dönüştürerek biçimlendirmiş oluyoruz. Başka bir deyişle, sayıyı temsil eden stringi biçimlendiriyoruz.

Çıktıyı biçimlendirmek için kullandığımız `String.Format()` metodu ile hemen hemen her kültüre uyan çıktı biçemleri elde edebiliriz.

Şimdi bu üç yöntemi örnekler üzerinde açıklamaya başlayalım.

`Write()` ya da `WriteLine()` metotları ile çıktı elde etme

En basit sayısal çıktıyı `System.Console` sınıfı içindeki `Write()` ya da `WriteLine()` metotları ile elde ederiz. Parametre yazarken tamsayıları olduğu gibi yazarız. Kesirli sayılarda kesir ayracı olarak (.) simgesini kullanarak yazarız. Konsola giden çıktı, o an bilgisayarı çalıştıran Windows İşletim Sisteminde etkin olan dil ve kültüre uyacak biçimde kendiliğinden biçimlenir. Tamsayılar olduğu gibi çıkar. Kesirli sayılar için girdi ile çıktı arasında yöresellik farkı oluşur. Türkçe Windows işletim sisteminde 1234.5678 biçiminde yazılan parametre, Türkçe'de kullandığımız 1234,5678 biçimiyle çıkar. Burada dikkat edilecek tek nokta, kesirli sayıları parametre olarak kullanılırken kesir ayracının (,) değil (.) olması gerektiğidir. Parametre girişi standarttır, ama konsol çıktısı dil ve kültüre göre değişir. Aşağıdaki program bir tamsayı ve bir kesirlisayı giriş ve çıkışını göstermektedir.

Biçem01.cs

```
using System;

namespace SayısalBiçemler
{
    class Sayılar01
    {
        static void Main(string[] args)
        {
            Console.WriteLine(12345678);
            Console.WriteLine(1234.5678);
        }
    }
}
```

Çıktı

```
12345678
1234,5678
```

Sayıları doğrudan parametre olarak kullanmak yerine, onları tutan değişken adlarını parametre olarak kullanabiliriz. Bu durumda da yukarıda söylediklerimiz geçerlidir.

Biçem02.cs

```
using System;

namespace SayısalBiçemler
{
    class Sayılar01
    {
        static void Main(string[] args)
        {
            int  tamSayı = 12345678;
            double kesirliSayı = 1234.5678;
            Console.WriteLine(tamSayı);
            Console.WriteLine(kesirliSayı);
        }
    }
}
```

Çıktı

```
12345678
1234,5678
```

Sağa/sola Yanaşık Yazdırma

string'lerde olduğu gibi tamsayıları belli bir alana sağa ya da sola yanaşık yazdırabiliriz. Bunun için `{}` `yer tutucusu`'nu kullanırız. `{n,xx}` yer tutucusundaki `n` sayısı 0,1,2,... sayılarından birisidir. Kaçıncı değişkene yer tutulacağını belirtir. `xx` pozitif tamsayısı değişken değerinin kaç haneye sağa yanaşık olarak yazılacağını belirtir. `-xx` sayısı değişken değerinin kaç haneye sola yanaşık olarak yazılacağını belirtir.

Biçem04.cs

```
using System;

namespace SayılarıBiçemleme
{
    class Sayılar
    {
        static void Main(string[] args)
        {
            Console.WriteLine("|{0,-20}|", 12345678);
            Console.WriteLine("|{0, 20}|", 12345678);
        }
    }
}
```

Çıktı

```
|12345678        |
|        12345678|
```

{} Yer Tutucu ile Biçemleme

`{ }` değişkene `yer tutucu`'dur. Bunun içine *basamak tutucuları* dediğimiz `'#'` ve `'0'` karakterlerini, sayıyı görmek istediğimiz biçemde yerleştiririz. `'#'` basmak tutucusu ancak sayıya gerektiği kadar basamak ayarlar. `'0'` basamak tutucusu ise, sayıda olmayan basamaklar yerine `'0'` koyar. Bunların işlevlerini 9807605,4361 sayısını farklı biçemlerde yazdırak görelim.

Biçem05.cs

```
using System;

public class ExponentialNotation
{
    public static void Main()
    {
        double dSayı = 9807605.4361;
        Console.WriteLine(dSayı);
        Console.WriteLine("{0}", dSayı);

        Console.WriteLine();
        Console.WriteLine("{0:0.00}", dSayı);
        Console.WriteLine("{0:#.##}", dSayı);

        Console.WriteLine();
        Console.WriteLine("{0:000,000,000.0000000}", dSayı);
        Console.WriteLine("{0:###,###,###.#######}", dSayı);

        Console.WriteLine();
        Console.WriteLine("{0:000000000.0000000}", dSayı);
        Console.WriteLine("{0:#########.#######}", dSayı);

        Console.WriteLine();
        Console.WriteLine("{0:##.##}", dSayı);
        Console.WriteLine("{0:00.00}", dSayı);

        Console.WriteLine();
        Console.WriteLine("{0:00.###}", dSayı);
        Console.WriteLine("{0:##.0}", dSayı);
    }
}
```

Çıktı

```
9807605,4361
9807605,4361

9807605,44
9807605,44

009.807.605,4361000
9.807.605,4361

009807605,4361000
9807605,4361
9807605,44
9807605,44
```

```
9807605,446
9807605,4
```

Bu çıktıyı satır satır incelersek, `'#'` ve `'0'` basamak tutucularının yaptıklarını hemen görebileceğiz.

1 ve 2 nci satırı veren deyimler, esasta aynıdırlar.

3 ve 4 üncü satırlarda, sayının tam kısmının basamak sayısı istenen biçemdekinden fazladır. Dolayısıyla, C# , isteğe uymaz, sayının tam kısmının basamaklarını eksiksiz yazar. Kesirli kısım iki haneli istenmiştir. Sayının kesir kısmı iki haneye yuvarlanır.

5 ve 6 ıncı satırlar için istenen biçemde, sayının tam ve kesirli basamakları sayısından fazla basamak tutucusu konmuştur. Bu durumda `'#' basamak tutucusu` gerektiği kadar haneyi kullandırır. `'0' basamak tutucusu` ise, boş kalan basamakları '0' karakteri ile doldurur.

7 ve 8 inci satırlar için istenen basamak sayıları 5-6 ıncı satırlardaki gibidir. Ancak, sayının tam kısmı binliklerine gruplanmamıştır. Dolayısıyla, çıktı, binlik gruplara ayrılmamış olarak gelmektedir.

9-10-11-12 inci satırlar için istenen biçemdeki tamsayı basamakları yetmediğinden, C# , o isteğe uymaz, sayının tam kısmını eksiksiz yazar. Kesirli kısım için istenen hane sayısı sayınınkinden az olduğundan, sayının kesir kısmı yuvarlanarak yazılır.

Standart Sayısal Biçem Belirtgenleri

Sayısal verileri biçimlendirmek için kullanılan metinler (strig) dir. Örneğin, finansal uygulamalarda para ifade eden sayıların önüne veya arkasına para biriminin yazılması ve çıktının kaç haneye yazılacağının belirtilmesi standart bir sayısal biçem belirtgenidir. `c12` belirtgeni sayının önüne para biriminin konulacağını ve çıktının 12 haneye sağa yanaşık yazılacağını belirtir. Belirgende ilk harf olarak `C,c, D,d, E,e, F,f, G,g, N,n, P,p, R,r, X,x` harflerinden birisi konulur. Bu harflerin işlevleri aşağıdaki tabloda belirtilmiştir.

Sayısal Biçem Belirtgenleri (Numeric Format Specifiers)

Belirteç	Biçem	Çıktı
C[n] ,c[n]	$XX,XX.XX ($XX,XXX.XX)	Para (Currency)
D[n]	[-]XXXXXXX	Sayısal (decimal)
E[n] , e[n]	[-]X.XXXXXXE+xxx [-]X.XXXXXXe+xxx [-]X.XXXXXXE-xxx [-]X.XXXXXXe-xxx	Üstel (Exponent)
F[n]	[-]XXXXXXX.XX	Kesir ayracı (Fixed point)
G[n]	General or scientific	Genel
N[n]	[-]XX,XXX.XX	Sayı
P[p]		Yüzde
R[r]		Dönüşlü (round-trip)
X[n] , x[n]	Hex representation	Hexadecimal

`R, r` : Bir sayının stringe dönüşmesi halinde tekrar aynı sayıya dönüşebileceğini garanti eder.

Aşağıdaki programlar tamsayılarda ve kesirli sayılarda sayısal biçim belirteçlerinin etkilerini göstermektedir. Satırların karşısında yazılan çıktıları dikkatle inceleyiniz.

Biçem06.cs

```
using System;
class TestDefaultBiçemler
{
    static void Main()
    {
        int i = 123456789;
        Console.WriteLine("{0:C}", i); // 123.456.789,00TL
        Console.WriteLine("{0:D}", i); // 123456789
        Console.WriteLine("{0:E}", i); // 1,2345678E+008
        Console.WriteLine("{0:F}", i); // 123456789,00
        Console.WriteLine("{0:G}", i); // 123456789
        Console.WriteLine("{0:N}", i); // 123.456.789,00
        Console.WriteLine("{0:X}", i); // 75BCD15
        Console.WriteLine("{0:x}", i); // 75bcd15,
    }
}
```

Biçem07.cs

```
using System;
class TestIntegerFormats
{
    static void Main()
    {
        int i = 123;
        Console.WriteLine("{0:C6}", i); // 123,000000TL
        Console.WriteLine("{0:D6}", i); // 000123
        Console.WriteLine("{0:E6}", i); // 1,230000E+002
        Console.WriteLine("{0:G6}", i); // 123
        Console.WriteLine("{0:N6}", i); // 123,000000
        Console.WriteLine("{0:X6}", i); // 00007B
        i = -123;
        Console.WriteLine("{0:C6}", i); // -123,000000TL
        Console.WriteLine("{0:D6}", i); // -000123
        Console.WriteLine("{0:E6}", i); // -1,230000E+002
        Console.WriteLine("{0:G6}", i); // -123
        Console.WriteLine("{0:N6}", i); // -123,000000
        Console.WriteLine("{0:X6}", i); // FFFFF85
        i = 0;
        Console.WriteLine("{0:C6}", i); // 0,000000TL
        Console.WriteLine("{0:D6}", i); // 000000
        Console.WriteLine("{0:E6}", i); // 0,000000E+000
        Console.WriteLine("{0:G6}", i); // 0
        Console.WriteLine("{0:N6}", i); // 0,000000
        Console.WriteLine("{0:X6}", i); // 000000
    }
}
```

Biçem08.cs

```
using System;
class TestDoubleFormats
{
    static void Main()
```

```
    {
        double d = 1.23;
        Console.WriteLine("{0:C6}", d); // 1,230000Tl
        Console.WriteLine("{0:E6}", d); // 1,230000E+000
        Console.WriteLine("{0:G6}", d); // 1,23
        Console.WriteLine("{0:N6}", d); // 1.230000
        d = -1.23;
        Console.WriteLine("{0:C6}", d); // -1,230000TL
        Console.WriteLine("{0:E6}", d); // -1,230000E+000
        Console.WriteLine("{0:G6}", d); // -1,23
        Console.WriteLine("{0:N6}", d); // -1,230000
        d = 0;
        Console.WriteLine("{0:C6}", d); // 0,000000TL
        Console.WriteLine("{0:E6}", d); // 0,000000E+000
        Console.WriteLine("{0:G6}", d); // 0
        Console.WriteLine("{0:N6}", d); // 0,000000
    }
}
```

Simgesel biçemler (picture formats)

Belirteç	Sonuç (string)	C# ifadesi
0	Sıfır basamak tutucu	Zero placeholder
#	Rakam basamak tutucu	Digit placeholder
.	Kesir ayracı	Decimal point
,	grup ayracı	Group separator or multiplier
%	Yüzde simgesi	Percent notation
E+0, E-0 e+0, e-0	Üstel notasyon	Exponent notation
\	Literal karakter belirtme	Literal character quote
'xx'"xx"	Literal string belirtme	Literal string quote
;	Bölüm ayracı	Section separator

Biçem09.cs

```
using System;
class TestIntegerCustomFormats
{
    static void Main()
    {
        int i = 123;
        Console.WriteLine("{0:#0}", i); // 123
        Console.WriteLine("{0:#0;(#0)}", i); // 123
        Console.WriteLine("{0:#0;(#0);<zero>}", i); // 123
        Console.WriteLine("{0:#%}", i); // 12300%
        i = -123;
        Console.WriteLine("{0:#0}", i); // -123
        Console.WriteLine("{0:#0;(#0)}", i); // (123)
        Console.WriteLine("{0:#0;(#0);<zero>}", i); // (123)
        Console.WriteLine("{0:#%}", i); // -12300%
        i = 0;
        Console.WriteLine("{0:#0}", i); // 0
        Console.WriteLine("{0:#0;(#0)}", i); // 0
```

```
        Console.WriteLine("{0:#0;(#0);<zero>}", i); // <zero>
        Console.WriteLine("{0:#%}", i); // %
    }
}
```

Alıştırmalar

Biçem10.cs

```
// Biçemleme örnekleri

using System;

public class PictureFormatDemo
{
    public static void Main()
    {
        double dSayı = 98765.10234;

        Console.WriteLine("Default biçem: " + dSayı);

        // 2 ondalık kesirli
        Console.WriteLine("İki kesirli hane: " +
                          "{0:#.##}", dSayı);

        // Binliklere ayır.
        Console.WriteLine("Binliklere ayrılmış: {0:#,###.##}", dSayı);

        // Bilimsel notasyon
        Console.WriteLine("Bilimsel notasyon: " +
                          "{0:#.###e+00}", dSayı);

        // Scale the value by 1000.
        Console.WriteLine("Value in 1,000s: " +
                          "{0:#0,}", dSayı);

        /* Pozitif, negatif ve sıfır
           sayılarının farklı gösterimleri */

        Console.WriteLine("Pozitif, negatif ve sıfır  sayılarının farklı
gösterimleri");
        Console.WriteLine("Pozitif sayı : {0:#.#;(#.##);0.00}", dSayı);
        dSayı = -dSayı;
        Console.WriteLine("Negatif Sayı: {0:#.##;(#.##);0.00}", dSayı);
        dSayı = 0.0;
        Console.WriteLine("Sıfır Sayısı: {0:#.##;(#.##);0.00}", dSayı);

        // Yüzde gösterimi.
        dSayı = 0.17;
        Console.WriteLine("Yüzde gösterimi: {0:#%}", dSayı);
    }
}
```

Çıktı

```
Default biçem: 98765,10234
İki kesirli hane: 98765,1
Binliklere ayrılmış: 98.765,1
```

```
Bilimsel notasyon: 9,877e+04
Value in 1,000s: 99
Pozitif, negatif ve sıfır  sayılarının farklı gösterimleri
Pozitif sayı : 98765,1
Negatif Sayı: (98765,1)
Sıfır Sayısı: 0,00
Yüzde gösterimi: 17%
```

Yer tutucuda sayının hanelerinden daha az ya da daha çok # hanesi yazılması, tamsayıyı değiştirmez. Çıktıda her zaman tamsayıya yetecek kadar hane ayrılır.

Biçem11.cs

```
// Biçemleme örnekleri

using System;

public class YerTutucuBiçemlemeleri
{
public class Resimleme
{
    public static void Main()
    {
        Console.WriteLine("{0:#####}", 123);
        Console.WriteLine("{0:#####}", 1234);
        Console.WriteLine("{0:###}", 12345);
    }
}
}
```

Çıktı

```
123
1234
12345
```

Çıktıda özellikle kesir haneleri istenmediğinde, kesirli sayı en yakın tamsayıya yuvarlanır.

Biçem12.cs

```
// Biçemleme örnekleri

using System;

public class YerTutucuBiçemlemeleri
{
public class Resimleme
{
    public static void Main()
    {
        Console.WriteLine("{0:#####}", 123.45678);
        Console.WriteLine("{0:#####}", 1234.56789);
        Console.WriteLine("{0:###}", 12345.908765);
    }
}
}
```

Çıktı

```
123
1235
12346
```

Çıkış biçeminde kaç tane kesirli sayı hanesi istediğimizi belirtebiliriz. İstenen hane sayısı sayının kesirli hanelerinden çok ise, boş kalan haneler sağa doğru 0 ile dolar. İstenen hane sayısı sayının kesirli hanelerinden az ise, sayının kesir kısmı yuvarlanarak alınır. Sayının tamsayı kısımları, istenen hane sayısından asla etkilenmez. Sayının tamsayı haneleri çıktıda aynen görünür.

Biçem13.cs

```
using System;

public class KesirAyracı
{
    public static void Main()
    {
        Console.WriteLine("{0:#####.000}", 98765.6);
        Console.WriteLine("{0:##.000}", 8765.654321);
    }
}
```

Çıktı

```
98765,600
8765,654
```

Üstel Notasyon

Her sayıyı üstel notasyonla yazabiliriz. Örneğin, `12345 = 1,2345x10^5` dir. Bu sayının konsola giden çıktısı `1,2345E+5` biçiminde gösterilir. `E+5` yerine `E5, E05, E005, E+05, E+005` gibi notasyonlar da kullanılır. Burada `E+5` ifadesi `(x10^5)` yerine geçer. E nin solunda kalan sayının 10 üssü 5 ile çarpılacağı anlamına gelir. E harfinden sonra gelen sayı `" 10 nun kuvveti"` veya `"10 nun üssü"` adını alır. Bu üs `bir, iki, üç, ...` haneli yazılabilir. Çünkü `5, 05, 005,...` sayıları aynıdır.

Üstel notasyonda, sayının tam kısmının kaç basamaklı olacağı, kesirli kısma kaç basamak konacağı, üstel sayının nasıl gösterileceği `{} yer tutucusu` içinde belirlenebilir. İstenen kesir basamaklarının sayısı, sayının kesir basamaklarından az değilse, kesir basamakları aynen gösterilir. İstenen kesir basamakları sayısı sayınınkinden az ise, sığmayan basamaktan sonrası yuvarlanır.

Biçem14.cs

```
"{0:#.000E-00}"
using System;

public class ÜstelNotasyon
{
    public static void Main()
    {
        double pi = 3.14159265358979323846264338327950288419716939937510;

        Console.WriteLine("{0:#.000E-00}", pi);
        Console.WriteLine("{0:0.000E-00}", pi);
        Console.WriteLine("{0:0.000E+00}", pi);
```

tSayı null

```
        Console.WriteLine("{0:#.0000000E+000}", pi);
        Console.WriteLine("{0:0.0000000E+000}", pi);
        Console.WriteLine("{0:#.0000000E-000}", pi);

        Console.WriteLine("{0:##.000E-00}", pi);
        Console.WriteLine("{0:#######.0000000E+000}", pi);
        Console.WriteLine("{0:#######.0000000E-000}", pi);
        Console.WriteLine("{0:#,###,###.0000000E+000}", pi);
    }
}
```

Çıktı

```
3,142E00
3,142E00
3,142E+00
3,1415927E+000
3,1415927E+000
3,1415927E000
31,416E-01
3141592,6535898E-006
3141592,6535898E-006
3.141.592,6535898E-006
```

`ToString()` metodunu kullanmak

C# dilinde 8 tane tamsayı sınıfı ile 3 tane kesirli sayı sınıfının var olduğunu söylemiştik. Bu sınıfların hepsinde `ToString()` metodu vardır. Bu metodun görevi, bir nesne olarak tutulan sayıyı istenilen biçeme (string) dönüştürmektir. Stringe dönüşen sayıyı, `Write()` veya `WriteLine()` metotları konsola gönderir.

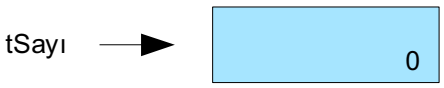
Aşağıdaki örnek `ToString()` metodunun kullanılışını göstermektedir. `ToString()` metodu bir tane değildir. Sayı sınıflarının hepsinde bir `ToString()` metodu vardır. Dolayısıyla, yazdıracağımız sayı hangi sınıfa aitse o sınıfın bir nesnesini yaratmamız gerekir. Başka bir deyişle, sayıyı o sınıfa ait bir nesne olarak yaratmak gerekir.

```
int tSayı ;
tSayı = new int();
tSayı = 12345678;
Console.WriteLine(tSayı.ToString());
```

deyimlerini bire birer inceleyelim.

```
int tSayı ;
```

deyimi, `int` sınıfına ait `tSayı` adlı bir nesne işaretçisinin bildirimidir. `tSayı`, int sınıfına ait bir nesnenin bellekteki adresini gösterecektir. Bu nedenle, ona işaretçi, referans, pointer sıfatlarını veririz. C/C++ dillerinde "pointer" sıfatı kullanılır. C# dilinde *"pointer"* sözcüğü yerine *"referans"* sözcüğü kullanılır. Biz *"referans"*, *"işaretçi"* ve *"pointer"* sıfatlarını eşanlamlı olarak kullanacağız. Henüz bu aşamada işaret edilecek nesne yaratılmamıştır. Dolayısıyla, `tsayı` işaretçisinin gösterdiği yer `'null'`dır; yani bellekte bir adres göstermez. Bir nesneyi işaret edebilmesi için, öncelikle o nesnenin yaratılmış olması gerekir. Bundan sonraki deyim onu yapmaktadır.




```
tSayı = new int();
```

deyimi int tipinden bir nesne yaratır; yani bellekte `int` tipi verinin sığacağı bir yer açar. Nesne Yönelimli programlamada bu eyleme nesne yaratma (constructor, instantiate) denir. Bellekte açılan o yere nesnenin adresi diyoruz. `tSayı` işaretçisi yaratılan nesnenin bellekteki adresini göstermeye başlar. Henüz bu aşamada o adrese bir değer girilmemiştir. C# sayısal nesneler yaratılır yaratılmaz onlara kendiliğinden 0 değerini (default value) atar. Bu değer nesneye başka değer atanınca değişir.

tSayı = 12345678;

deyimi, yaratılan nesneye 12345678 değerini atar. Bu eylem, 12345678 sayısının açılan adrese girmesi demektir. O sayı artık bir `int` nesnesidir. `int` sınıfına ait bütün özelikleri taşır, ona `int` sınıfına ait metotlar uygulanabilir.

Dördüncü satırdaki

```
Console.WriteLine(tSayı.ToString());
```

deyimi ard arda iki iş yapmaktadır. `tSayı.ToString()` metodu `tSayı` işaretçisinin işaret ettiği nesneyi stringe dönüştürür. `WriteLine()` metodu onu konsola yazar.

Benzer işleri kesirli sayılar için de yaparız. Aşağıdaki kodlar o işi yapmaktadır.

```
double kSayı = new double();
kSayı = 1234.5678;
Console.WriteLine(kSayı.ToString());
```

Yukarıda iki satırda yapılan iş burada ilk satıra tek deyim olarak yazılmıştır. Bu deyim double sınıfına ait bir nesnenin bellekteki adresini gösterecek kSayı işaretçisinin bildirimini yapmakta ve onun göstereceği nesneye bellekte bir yer ayırmaktadır. İkinci ve üçüncü satırın işlevleri yukarıda açıkladıklarımıza benzer.

Bütün bu söylediklerimizi aşağıdaki programa koyalım.

Biçem15.cs

```
using System;

namespace SayılarıBiçemleme
{
    class Sayılar
    {
        static void Main(string[] args)
        {
            int tSayı ;
            tSayı = new int();
            tSayı = 12345678;
            Console.WriteLine(tSayı.ToString());

            double kSayı = new double();
            kSayı = 1234.5678;
            Console.WriteLine(kSayı.ToString());
        }
    }
}
```

Çıktı

```
12345678
1234,5678
```

1.

Biçem16.cs

```
using System;

namespace SayılarıBiçemleme
{
    public class ToStringBiçemi
    {
        public static void Main()
        {
            int n = 123;
            double kks = 0.35;
            double bks = 12345.67809;

            string str = bks.ToString("F2");
            Console.WriteLine(str);

            str = bks.ToString("N5");
            Console.WriteLine(str);

            str = bks.ToString("e");
            Console.WriteLine(str);

            str = bks.ToString("r");
            Console.WriteLine(str);

            str = kks.ToString("p");
            Console.WriteLine(str);

            str = n.ToString("X");
            Console.WriteLine(str);

            str = n.ToString("D12");
            Console.WriteLine(str);

            str = 189.99.ToString("C");
            Console.WriteLine(str);
        }
    }
}
```

Çıktı

```
12345,68
12.345,67809
1,234568e+004
12345,67809
%35,00
7B
000000000123
189,99 TL
```

NumberFormatInfo

Bu sınıf, kültürlerden bağımsız biçemler yaratır. Biçemleme seçeneklerini değiştirebilir. Örneğin, pozitif sayıların, yüzde simgesinin, binliklere ayırma ayracının, para biriminin nasıl gösterileceği gibi şeyleri onunla yapabiliriz.

Bu söylediklerimizin nasıl yapıldığını örnekler üzerinde açıklayacağız. Aşağıdaki ilk örnekte 1234567890 sayısının çıktısı *Türkçe, Fransızca* ve *Almanca* biçemleriyle verilmektedir.

Biçem17.cs

```
using System;
using System.Globalization;
using System.Threading;

namespace Yöresellik
{
    class Yerellik01
    {
        static void Main(string[] args)
        {
            int birSayı = 1234567890;

            // mevsut thread'deki yerel (Türkçe)biçim
            Console.WriteLine(birSayı.ToString("N"));

            // IFormatProvider kullanımı ile Fransızca
            Console.WriteLine(birSayı.ToString("N",
                              new CultureInfo("fr-FR")));

            // kültür thread'inin değiştirilmesi (Almanca)
            Thread.CurrentThread.CurrentCulture =

                              new CultureInfo("de-DE");
            Console.WriteLine(birSayı.ToString("N"));
        }
    }
}
```

Çıktı

```
1.234.567.890,00
1 234 567 890,00
1.234.567.890,00
```

İlk satırı yazan

```
Console.WriteLine(birSayı.ToString("N"));
```

deyimidir.

```
birSayı.ToString("N")
```

metodu `birSayı` nesnesini `N biçemine` dönüştürerek standart çıkışa (konsol) gönderiyor. N biçemi, o anda etkin olan kültür thread'ine göre sayının yazılış biçimidir. Bilgisayarımız Türkçe Windows işletim sistemiyle çalıştığı için, doğal (default) kültür thread'i Türkçe'dir. Dolayısıyla, programdaki ilk `ToString("N")` metodu `birSayı` nesnesini (ki o 1234567890 sayısıdır) Türkçe biçemli string'e dönüştürecektir. `Console.WriteLine()` metodu ise, o dönüşmüş string'i konsola (standart çıkış) yazar.

İkinci satırı yazan

```
Console.WriteLine(birSayı.ToString("N", new CultureInfo("fr-FR")));
```

deyimidir. Burda ToString() metodunun ikinci parametresi

```
new CultureInfo("fr-FR")
```

deyimi ile yaratılan CultureInfo sınıfına ait bir nesnedir. Bu nesne "fr-FR" parametresi ile Fransız kültür thread'ini almış olur. Dolayısıyla,

```
ToString("N", new CultureInfo("fr-FR"))
```

metodu birSayı nesnesini Fransızca'daki sayı yazma biçemiyle string'e dönüştürür.

Üçüncü satır da benzer olarak Almanca sayı yazma biçemine dönüşmektedir.

Biçem18.cs

```
// String.Format() kullanımına örnek
using System;

namespace SayılarıBiçemleme
{
    public class FormatDemo2
    {
        public static void Main()
        {
            int i;
            int toplam = 0;
            int çarpım = 1;
            string str;

            /* Program koşarken 1-10 arasındaki sayıların
               toplmı ve çarpımını listeler */
            for (i = 1; i <= 10; i++)
            {
                toplam += i;
                çarpım *= i;
                str = String.Format("Toplam:{0,3:D}  Çarpım:{1,12:D}",
                                    toplam, çarpım);
                Console.WriteLine(str);
            }
        }
    }
}
```

Çıktı

```
Toplam:  1  Çarpım:          1
Toplam:  3  Çarpım:          2
Toplam:  6  Çarpım:          6
Toplam: 10  Çarpım:         24
Toplam: 15  Çarpım:        120
Toplam: 21  Çarpım:        720
Toplam: 28  Çarpım:       5040
Toplam: 36  Çarpım:      40320
Toplam: 45  Çarpım:     362880
Toplam: 55  Çarpım:    3628800
```

Sorular

1. Sayıları 5 haneye ön boşluklara 0 lar koyarak nasıl yazdırılır?

```
int _num1 = 123;
int _num2 = 45;
int _num3 = 123456;
String.Format("{0:00000}", _num1); //"00123"
       String.Format("{0:00000}",   _num2); //"00045"
       String.Format("{0:00000}", _num3); //"123456"
String.Format("{0:d5}", _num1); //"00123"
       String.Format("{0:d5}", _num2);   //"00045"
       String.Format("{0:d5}", _num3); //"123456"
```

2. n=0 için "Evet" onun dışındakiler için "Hayır" yazdıran bir deyim yazınız ?

```
Console.WriteLine(String.Format("{0:Hayır;;Evet}",  n));
```

Çıktı: n=0 ise *"Evet"*, değilse *"Hayır"* yazar.

3. Bir sayıyı string tipine dönüştüren şu kodlardan hangisi tercih edilmelidir?

```
Double testDouble = 17.73;
    // boxing yapar
String testString1 = String.Format("{0:C}", testDouble);
    // boxing yapmaz
String testString2 = testDouble.ToString("C");
```

4. Bir konsol çıktısına { } parantezleri yazdırılabilir mi?

Evet.

```
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(String.Format("{{21.yy
cahilleri}}={0}","Unutmayı ve yeniden Öğrenmeyi bilmeyenler"));
        }
    }
```

5. 03122341010 biçiminde verilen telefon numarasını 0312-234 10 10 biçiminde nasıl yazdırabilirsiniz?

```
class Program
    {
        static void Main(string[] args)
        {
            string str = String.Format("{0:0###-### ## ##}",
03122341010);
            Console.WriteLine(str);
        }
    }
```

Bölüm 27

string'den sayıya dönüşüm

Convert Sınıfı
Parse() Metodu
Biçemli Stringe Dönüşen Sayıların Ters Dönüşümü
CultureInfo sınıfı
Sayı Stilleri Listesi (Number Styles)

Önceki bölümlerde sayıları istenen biçemde stringe dönüştürmeyi öğrendik. Programcılıkta, bu işin tersine de sık sık gerekseme duyarız. Herhangi bir biçemde stringe dönüştürülmüş sayıyı tekrar sayının aslına dönüştürmek gerekebilir. Bu işi yapacak yöntemlere sahibiz.

String tipinin giriş/çıkış işlemlerinde yaşamsal rol oynadığını biliyoruz. Bilgisayara girişlerin ve çıkışların hemen hepsi string tipindendir. Örneğin, klavyeden yaptığımız bütün girişler string tipindendir. Sistem giriş biriminden aldığı stringi içeride olması gereken veri tipine dönüştürür. Bunun tersi de doğrudur. Örneğin, bilgisayardan gelen bir sayısal çıktı, konsola veya yazıcıya rakamlardan oluşan bir string olarak gider. Çıkışta sayıdan stringe dönüşüm gene sistem tarafından yapılır. Burada sistem derken, işletim sistemi derleyici ve kaynak programdan oluşan bütünü kasdediyoruz.

Öyleyse girdi/çıktı işlemlerinde bir veri tipinden başka bir veri tipine dönüşüm söz konusudur. Daha önce sayısal veri tiplerinin birinden ötekine *istemsiz* (implicit) veya *istemli* (explicit) dönüşümler (casting) yapmıştık. Ama şimdi sözünü ettiğimiz dönüşüm bir 'casting' olayı değildir. Ondan çok farklıdır. Bu dönüşüm sayıdan sayıya değil, sayıdan stringe ve stringden sayıya yapılan ters dönüşümlerdir.

Özellikle şunu da belirtmeliyiz. Sayılar ile stringler arasındaki bu dönüşüm, başka veri tiplerine uygulanamaz.

Convert sınıfı

`.NET Framework` System namespace içinde `Convert` adlı bir sınıf sunar. Bu sınıf çok sayıda static dönüşüm metodu içerir. Metotlar `static` olduğu için, onlar sınıfa ait bir nesne yaratmadan doğrudan kullanılabilirler. Böyle oluşu, programcının işini kolaylaştırır. Dönüşümler için, C# sayısal sınıfları ile .NET sınıfları arasındaki ilişkiyi anımsayalım.

C# Veri Tipi	.NET Yapı Adı
bool	Boolean
byte	Byte
decimal	Decimal
double	Double
float	Single
int	Int32
long	Int64
sbyte	SByte
short	Int16
uint	UInt32
ulong	UInt64
ushort	UInt16

Convert sınıfının metotlarından bazıları şunlardır:

```
ToBoolean, ToByte, ToChar, ToDateTime, ToDecimal, ToDouble, ToInt16,
ToInt32, ToInt64, ToSByte, ToSingle, ToString, ToUInt16, ToUInt32,
ToUInt64
```

Bu metotların işlevleri adlarından bellidir. Örneğin,

Convert.ToInt32	metodu stringe dönüşmüş tamsayıyı, tamsayı aslına dönüştürür (Int32).
Convert.ToDecimal	metodu stringe dönüşmüş kesirlisayıyı, kesirlisayı aslına dönüştürür.
Convert.ToBoolean	metodu "true" ve "false" stringlerini, sırasıyla True ve False boolean değerlerine dönüştürür. "true" ve "false" stringlerinin büyük ya da küçük harfle yazılmış olmaları sonuca etkimez.

```
using System;
class TestIntegerCustomFormats
{
```

```
    static void Main()
    {
        // Stringi tamsayıya dönüştür
        int stringToInt = Convert.ToInt32("12345");
        Console.WriteLine(stringToInt); // Çıktı: 12345

        // Stringi kesirli sayıya dönüştür
        decimal stringToDecimal = Convert.ToDecimal("12345,6789");
        Console.WriteLine(stringToDecimal);  // Çıktı : 12345,6789

        /* "true" stringini boolean'a dönüştür
           "true" nun büyük/küçük harfle yazılması
           sonucu etkilemez
       */
        bool stringToTrue = Convert.ToBoolean("true");
        Console.WriteLine(stringToTrue); // Çıktı : True

        /* "false" stringini boolean'a dönüştür
           "false" 'in büyük/küçük harfle yazılması
           sonucu etkilemez
       */
        bool stringToFalse = Convert.ToBoolean("FALse");
        Console.WriteLine(stringToFalse); // Çıktı : False
    }
}
```

Kesirli sayıları tamsayılara dönüştürmek için `casting` yerine `ToInt32()` metodu kullanılabilir. Yalnız bir farklılığa dikkat etmek gerekir. Casting sayının kesir kısmını atar, tamsayı kısmını alır. ToInt32() metodu kesirli sayıyı en yakın tamsayıya yuvarlar. Aşağıdaki program bu ayrımı göstermektedir.

```
using System;
class Dönüşümler
{
    static void Main()
    {
        decimal ilkSayı = 17.5M;
        Console.WriteLine(ilkSayı);  // Çıktı: 17,5
        int tamSayıya = Convert.ToInt32(ilkSayı);
        Console.WriteLine(tamSayıya); // Çıktı: 18
        int castEdilen = (int)ilkSayı;
        Console.WriteLine(castEdilen); // Çıktı: 17
    }
```

Parse() Metodu

`Parse()` metodu `string` olarak yazılan sayıları aslına dönüştürür. Bir bakıma `ToString()` metodunun tersidir.

Bütün sayı sınıflarının birer tane `ToString()` metodu olduğunu ve bunların herbirinin dörder tane overload edilmiş sürümlerinin varlığını biliyoruz. Bunları kullanarak, sayıları istediğimiz biçemde stringe dönüştürebiliyoruz.

Benzer olarak bütün sayı sınıflarının birer tane `Parse()` metodu vardır. Bunların da *aşkın yüklenmiş* (`overloaded`) dörder tane sürümleri vardır.

`ToString()` metodunun yaptığı işin tersini yapan `Parse()` metodunun, en azından `ToString()` metodu ile stringe dönüşen sayıları, bir ters dönüşüm gibi, kendi özlerine dönüştürmesini umut etmeliyiz. Gerçekten, `Parse()` metodu bu ters dönüşümleri yapabilecek yeteneklere sahiptir. Bu yeteneklerini, aşağıda bazı örnekler üzerinde göreceğiz.

Parse Metotları

- Parse (String)
- Parse (String, NumberStyles)
- Parse (String, IFormatProvider)
- Parse (String, NumberStyles, IFormatProvider)

Birincisini kullanmayı daha ilk bölümlerde öğrendik. Stringe dönüşmüş `int` tipi bir tamsayıyı kendi özüne dönüştürmek için

```
int.Parse(s);
```

dönüşümünü kullanırız. Aşağıdaki program bunun nasıl yapıldığını göstermektedir.

Parse01.cs

```
using System;

namespace  StringdenSayıyaDönüşüm
{
    class Parse01
    {
        static void Main(string[] args)
        {
            string s = "12345";
            int n = int.Parse(s);
            n++;
            Console.WriteLine(n);// Çıktı: 12346
        }
    }
}
```

Aşağıdaki program 123 sayısını önce `ToString()` metodu ile "123" stringine, sonra `Parse()` metodu ile tekrar 123 sayısına dönüştürüyor.

IntParse.cs

```
using System;
using System.Globalization;
using System.Threading;

namespace Dönüşümler
{
    class IntParse
```

```
    {
        static void Main(string[] args)
        {
            int n = 123;
            string s = n.ToString();     // s = "123"
            n = Int32.Parse(s);          // n =  123
            Console.WriteLine(n);        // 123
        }
    }
}
```

Biçemli Stringe Dönüşen Sayıların Ters Dönüşümü

Sayının “ $12.345,6789” biçiminde stringe dönüşmüşünü bizler hemen algılarız. Ama bilgisayarın bunu algılamasını umamayız. Ona, $ işaretinin önünde ve ardında boşluklar olduğunu, sayının binlik gruplara ayrıldığını ve kesir ayracının var olduğunu bildirmeliyiz. `Parse()` metodu o zaman stringi algılar ve onu sayının aslına geri dönüştürebilir. `Parse()` metodu bu iş için yeterince araçlara sahiptir. Böyle durumlarda `Parse()` metodunun öteki sürümlerini kullanırız. O metotların nasıl kullanıldığıa dair birkaç örnek vereceğiz.

Aşağıdaki program, binlik basamaklarına ayrılmış bir sayıyı çözümleyip aslına dönüştürüyor.

DoubleParse01.cs

```
using System;
using System.Globalization;
using System.Threading;

namespace Metotlar
{
    class DoubleParse01
    {
        static void Main(string[] args)
        {
            string s = "12.345.678" ;

            // Binlik gruplarına ayrılmış sayıyı aslına dönüştür
            int n = int.Parse(s, NumberStyles.AllowThousands); //
12345678
            Console.WriteLine(n);  //Çıktı: 12345678
        }
    }
}
```

Aşağıdaki program, kesir ayracı içeren bir stringi çözümleyip asıl sayıya dönüştürüyor.

DoubleParse.cs

```
using System;
using System.Globalization;
using System.Threading;
```

```
namespace Metotlar
{
    class DoubleParse
    {
        static void Main(string[] args)
        {
            string s = "12345,678" ;

            // Kesir ayracı içeren stringi asıl sayıya dönüştür
            double flt = double.Parse(s, NumberStyles.AllowDecimalPoint);
            Console.WriteLine(flt); // Çıktı: 12345,678
        }
    }
}
```

Aşağıdaki program TL para birimi simgesini içeren stringi çözümleyip asıl sayıya dönüştürüyor.

Cparse.cs

```
using System;
using System.Globalization;
using System.Threading;

namespace Metotlar
{
    class cParse
    {
        static void Main(string[] args)
        {
            string s = "12345TL" ;

            // TL para birimi içeren stringi
            // asıl sayıya dönüştürüyor
            int n = int.Parse(s, NumberStyles.AllowCurrencySymbol);
            Console.WriteLine(n);  // Çıktı: 12345
        }
    }
}
```

Bazan string birden çok biçemleme öğesi içerebilir. Örneğin, “12345,67TL” stringi hem para birimini hem de kesir ayracı içeriyor. Bu durumda `OR` ya da `(|)` simgesini kullanarak her ikisini çözümlenmesini sağlayabiliriz. Aşağıdaki program o işi yapmaktadır.

ckParse.cs

```
using System;
using System.Globalization;
using System.Threading;

namespace Metotlar
{
    class ckParse
    {
        static void Main(string[] args)
        {
            string s = "12345,67TL";
```

```
            // Kesir ayracı ve para birimi içeren stringi
            // çözümleyip asıl sayıya dönüştürür
            float flt = float.Parse(s,
              NumberStyles.AllowDecimalPoint |
NumberStyles.AllowCurrencySymbol);
            Console.WriteLine(flt); // Çıktı : 12345,67
        }
    }
}
```

`CultureInfo` sınıfı

Farklı kültürlerde kesirli sayıların yazılışları farklıdır. Örneğin, ABD ve bazı avrupa ülkeleri kesirli sayı yazarken, ondalık kesir ayracı olarak nokta (.) ve yüzlükleri gruplamak için virgül (,) kullanır. Türkiye ve başka bazı ülkeler bunun tam tersini yaparlar; ondalık kesir ayracı olarak virgül (,) yüzlükleri gruplamak için nokta (.) kullanır. Dolayısıyla, bu farklı kültürlerde, örneğin 123.456 sayısı çok farklı anlamlara sahiptir. Bu nedenle, birçok bilgisayar programının yaptığı gibi, C# dili de bu ayrımı yapacak yeteneğe sahiptir. Programcının hangi kültüre göre sayıyı yazdırmak istediğini doğru sözdizimiyle ifade etmesi yeterlidir. Bunun için `CultureInfo` sınıfı kullanılır.

Aşağıdaki program `s="123,456"` biçimindeki bir stringi ABD sistemine göre tamsayıya çevirmek istemektedir. Ancak derleyici string içindeki (,) ün hangi sisteme göre yazıldığına karar veremez; verirse istenmedik yanlışlar doğar. O nedenle, hata iletisi verir.

Parse02.cs

```
using System;
using System.Globalization;

namespace Methods
{
    class Parsing
    {
        static void Main(string[] args)
        {
            CultureInfo vCultureInfo = new CultureInfo("en-US");
            string s = "123,456";
            int n = int.Parse(s, vCultureInfo);
            Console.WriteLine(n);
            // System.Format hatası.
        }
    }
}
```

Hata iletisi:

```
Unhandled Exception: System.FormatException: Input string was not in a correct format.
  at System.Number.StringToNumber(String str, NumberStyles options, NumberBuffer& number, NumberFormatInfo info,
Boolean parseDecimal)
  at System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInfo info)
  at Methods.Parsing.Main(String[] args) in C:\vsProjects\Metotlar\Metotlar\Parse02.cs:line 14
```

stringi sayıya çevirirken, programcının string içindeki virgülün (,) yüzlükleri ayıran ayraç olduğunu özenle belirtmesi gerekir. Bunun için

```
int n = int.Parse(s, vCultureInfo);
```

deyiminin yerine

```
int n = int.Parse(s, NumberStyles.AllowThousands, vCultureInfo);
```

deyimini yazmak yetecektir. Deneyerek sonucu görünüz.

Alıştırmalar

Aşağıdaki iki program, farklı stringleri çözümleyip sayıya dönüştürmektedir. Programı satır satır inceleyip, deyimlerin yaptığı işi anlayınız.

ParseMetodu

```
using System;
using System.Text;
using System.Globalization;

public sealed class ParseMetodu
{
    static void Main()
    {
        // stringi hex imiş gibi çözümle, decimal olarak yaz
        String num = "b";
        int val = int.Parse(num, NumberStyles.HexNumber);
        Console.WriteLine("{0} hex = {1}  decimal.", num, val);

        // işaret öntakısını algılayarak ama tekrarlayan
        //boşlukları ihmal ederek stringi çözümle
        num = "    -72   ";
        val = int.Parse(num, NumberStyles.AllowLeadingSign |
            NumberStyles.AllowLeadingWhite |
NumberStyles.AllowTrailingWhite);
        Console.WriteLine("'{0}' çözümlenince '{1}' oldu.", num, val);

        // parantezleri algılayarak ama tekrarlayan boşlukları
        //ihmal ederk stringi çözümle
        num = "    (86)   ";
        val = int.Parse(num, NumberStyles.AllowParentheses |
NumberStyles.AllowLeadingSign | NumberStyles.AllowLeadingWhite |
NumberStyles.AllowTrailingWhite);
        Console.WriteLine("'{0}' çözümlenince '{1}' oldu.", num, val);
    }
}
```

Çıktı

```
b hex = 11  decimal.
'   -72   ' çözümlenince '-72' oldu.
```

```
'   (86)  ' çözümlenince '-86' oldu.
```

```
using System;
using System.Globalization;
using System.Threading;

namespace Metotlar
{
    class DoubleParse
    {
        static void Main(string[] args)
        {
            // en-US kültür threadini etkin yap.
            Thread.CurrentThread.CurrentCulture =
CultureInfo.CreateSpecificCulture("en-US");

            string value;
            NumberStyles styles;

            // Yalnızca AllowExponent flagı ile üstel notasyonu çözümle
            value = "-5.072E+02";
            styles = NumberStyles.AllowExponent;
            ShowNumericValue(value, styles);

            // AllowExponent flagı ile Number flagı
            // kullanarak üstel notasyonu çözümle
            styles = NumberStyles.AllowExponent | NumberStyles.Number;
            ShowNumericValue(value, styles);

            // $ simgesi önünde ve arkasında tekrarlanan
            // boşluklar olan stringi çözümle
            value = " $ 3,841.7561  ";
            styles = NumberStyles.Number |
NumberStyles.AllowCurrencySymbol;
            ShowNumericValue(value, styles);

            // Binlik gruplarına ayrılmış ve kesir ayracı olan stringi
çözümle
            value = "(8,245.73)";
            styles = NumberStyles.AllowParentheses |
NumberStyles.AllowTrailingSign |
                     NumberStyles.Float;
            ShowNumericValue(value, styles);

            styles = NumberStyles.AllowParentheses |
NumberStyles.AllowTrailingSign |
                     NumberStyles.Float | NumberStyles.AllowThousands;
            ShowNumericValue(value, styles);
        }


        private static void ShowNumericValue(string value, NumberStyles
styles)
        {
            double number;
            try
            {
```

```
                number = Double.Parse(value, styles);
                Console.WriteLine("Converted '{0}' using {1} to {2}.",
                                  value, styles.ToString(), number);
            }
            catch (FormatException)
            {
                Console.WriteLine("Unable to parse '{0}' with styles
{1}.",
                                  value, styles.ToString());
            }
            Console.WriteLine();
        }
    }
}
```

Çıktı

```
/Unable to parse '-5.072E+02' with styles AllowExponent.
Converted '-5.072E+02' using AllowTrailingSign, AllowThousands, Float to -507.2.
Converted ' $ 3,841.7561  ' using Number, AllowCurrencySymbol to 3841.7561.
Unable to parse '(8,245.73)' with styles AllowTrailingSign, AllowParentheses, Float.
Converted '(8,245.73)' using AllowTrailingSign, AllowParentheses, AllowThousands
, Float to -8245.73.
```

Sayı Stilleri Listesi (Number Styles) :

C# Sayı Stili	Açıklama	Örnek
AllowCurrencySymbol	String para birimi simgesi içerebilir	"15000TL"
AllowDecimalPoint	String bir tek kesir ayracı içerebilir	"15000,00"
AllowExponent	String üstel biçimde olabilir	"1.5E3"
AllowHexSpecifier	String yalnızca hex biçimindedir	"5DC"
AllowLeadingSign	Stringin önünde + veya – işareti olabilir	"-15000"
AllowLeadingWhite	Stringin önünde boş karakter ya da kontrol karakterleri olabilir (Unicode characters 9 to 13).	" 15000"
AllowParentheses	String () içindeyse, sayıyı negatif sayı yap	"(15000)"
AllowThousands	String binlik gruplara ayrılabilir	"1.5000"
AllowTrailingSign	Stringin sonunda + veya – simgesi olabilir	"15000-"
AllowTrailingWhite	String tekrarlanan boş karakter içerebilir	"15000 "

Any	Yukarıdaki "Allow" stillerinin karması olabilir (hexedecimal hariç)	"1.5E3-"
Currency	Hex ve üstel dışındaki stringler, yukarıdakilerin bileşimi biçiminde olabilir	"(£1,5000.00)"
Float	String AllowDecimalPoint, AllowExponent, AllowLeadingSign, AllowLeadingWhite ve AllowTrailingWhite bileşimi olabilir	"-1,5000.00"
HexNumber	String AllowHexSpecifier, AllowLeadingWhite ve AllowTrailingWhite bileşimi olabilir	" 5DC "
Integer	String AllowLeadingSign, AllowLeadingWhite ve AllowTrailingWhite bileşimi olabilir	" -15000"
None	String hiç bir biçem içermiyor olabilir	"15000"
Number	String AllowDecimalPoint, AllowLeadingSign, AllowLeadingWhite, AllowThousands, AllowTrailingSign ve AllowTrailingWhite bileşimi olabilir.	" -1,5000.00

Kültürler ve Bölgeler

Farklı ülke ve kültürlerde sayıların yazılışı, tarih'in yazılışı, alfabetik sıralamanın yapılışı gibi eylemler birbirlerinden farklıdır. Hatta bazı alfabelerde yazı sağdan sola doğru yazılır. Farklı ülkeler farklı takvimler kullanır. Örneğin, `GregorianCalendar`, `HebrewCalendar`, `JapaneseCalendar`. C# dili farklı kültürlerin 'locale' (yöresel veya yerel) denilen farklı gereksemelerine yanıt verebilme yeteneğine sahiptir. Bu yeteneğini, `System.Globalization` aduzayı (namespace) ile yapmaktadır. Bu aduzayı içinde, bütün yörelerin isteklerine yanıt verebilecek sınıflar ve sınıflar içinde özelikler ve metotlar vardır. Çok geniş bir alanı dolduran bu konuyu sistematik incelemeye gitmeyeceğiz. Örneklerle, tarih ve sayı yazdırmada yöresel kullanışlara nasıl geçilebildiğini anlatmakla yetineceğiz. Konuyu sistematik incelemek isteyenler msdn web sitesinde .NET kütüphanesine bakabilirler.

Namespace `System.Globalization`

Dünyamız farklı bölgelere ve farklı kültürlere ayrılmıştır. `System.Globalization` namespace içinde bu farklılıkları gözetecek yaklaşık 30 sınıf vardır. Programcı gerekseme duyduğunda onları msdn web sitesinden görebilir. Konuyu çok dağıtmamak için bu aduzayının sistematik incelemesine girmeyeceğiz. Ancak, şimdiye kadar yaptığımız öneklere ek olarak, bundan sonraki bölümde, çeşitli kültürlere göre tarih yazdırma yöntemlerini görecek ve yeterince örnek yapacağız.

Class CultureInfo

`System.Globalization` namespace içinde bir sınıftır. Yöresel (locale) diye adlandırılan kültüre dayalı bilgileri tutar. Kültürün adı, yazı sistemi, kullanılan takvim, tarih'in yazılış biçemi, string'lerin sıralanma (sorting) yöntemi gibi şeyler tutulan bilgiler arasındadır. Dil, alt-dil grubu, ülke/bölge, takvim gibi kültüre dayalı biçemleri o yöre halklarının alıştığı gibi kullanmak elbette çok önemlidir. Farklı ülke ve kültürlerde kesirli sayıların yazılışı, tarihin yazılışı, alfabetik sıralamanın yapılışı gibi eylemler birbirlerinden farklıdır. Bazı alfabelerde yazı sağdan sola doğru yazılır. Örnekse, Arapça'nın 20 ye yakın ülke/bölge farklılığı, İngilizce'nin 10 dan fazla ülke/bölge farklılığı vardır. `CultureInfo` sınıfı bize bu bilgileri verir.

Aşağıdaki kısaltılmış tablo bu konuda bir fikir verebilir. Tablonun tamamı için msdn web sitesine bakılabilir.

Kısa Adı	Kodu	Dil
" " Boş string	0x007F	Invariant culture
ar	0x0001	Arabic
ar-EG	0x0C01	Arabic (Egypt)
en	0x0009	English
en-GB	0x0809	English (United Kingdom)
en-US	0x0409	English (United States)
es	0x000A	Spanish
es-VE	0x200A	Spanish-Venezuela
de	0x0007	German
de-AT	0x0C07	German-Austria
tr	0x001F	Turkish
tr-TR	0x041F	Turkish (Turkey)

CultureAndRegionInfoBuilder Sınıfı

Bir kültüre dayalı alt kültürleri belirler, yenilerini tanımlar. Gereksiz yer kaplayacakları için, bütün alt-kültürlere ait bilgiler çalışan her işletim sisteminde faal değildir. Kullanılmak istenen alt kültür sisteme yüklenebilir.

Thread Sınıfı

`System.Threading` aduzayı içinde bir sınıftır. Thread yaratır ve kontrol eder, önceliğini belirler, statüsünü tutar.

.NET birden çok işi eşzamanlı yapabilir. Farklı işlerin birbirleriyle karışmadan farklı yollardan işleme girdiğini düşününüz. Bu yollara `thread` denir.

Bölüm 28

Tarih İşleme ve Yazdırma Biçemleri

Bilgisayar uygulamalarında karakterler ve sayılar gibi önem taşıyan başka bir kavram tarih ve zamandır. Yaşantımızın her anı bir şekilde tarih ve zamanla iç içedir. O nedenle, tarih ve zamanla ilgili pek çok işlem yaparız. Tarihleri karşılaştırırız, birbirinden çıkarırız, bir zamana başka bir zaman dilimini ekleriz veya çıkarırız. İleriye veya geriye doğru bir zamanın bileşenlerini bilmek isteriz. Bu istekler, haftanın hangi günü doğduğumuzu doğum tarihimizden çıkarmak gibi basit bir merak olabileceği gibi, bir iş adamının 180 gün sonra ödeyeceği yüklü bir bononun hangi tarihe rasladığını bilmek gibi finansal bir konu da olabilir. Ayrıca, farklı meridyenlerdeki zamanlar farklıdır. Onların birisindeki zamanı bir başkasındaki zamana dönüştürmek gerekir. Farklı ülke ve kültürlerde tarih ve zamanı yazma biçemi farklı olduğu gibi, bir ülkede bile tarih ve zamanı yazmak için birden çok biçem kullanılır. Dolayısıyla, bilgisayar programları bütün bu sorunların üstesinden gelmek zorundadır. C# dili tarih ve zamanla ilgili her işlemi yapabilir, her dil ve kültürdeki farklı biçemlerde tarih yazabilir.

Bu bölümde, bunların nasıl yapıldığını örneklerle anlatacağız.

DateTime Yapısı

.NET Framework Class kütüphnesinde `System` namespace içindeki `DateTime` yapısı (structure) tarih ve zaman ile ilgili bilgileri tutan ve tuttuğu bilgileri istenen biçimde işleyen metotlara sahiptir. Özgenler (property) dediğimiz bu öğeler yıl, ay, gün, saat, dakika, saniye, milisaniye, vb. cinsinden tarih ve zamanı belirlemeye yarar. Metotlar ise, tarih ve zaman ile ilgili her türlü işlemi yapar ve istenen biçemde standart çıkışa gönderir.

DateTime Yapısının Özgenleri (properties)

Date, Day, DayOfWeek, DayOfYear, Hour, Kind, Millisecond, Minute, Month, Now, Second, Ticks, TimeOfDay, Today, UtcNow, Year.

Adlarının ima ettiği gibi, bu özgenlerin her birisi bir **`an (bir anlık zaman)`**'ın belli bir bileşenini tutar. Örneğin, `Date` o anın tarihini, `Day` o anın ayın kaçıncı günü olduğunu, `Month` o anın hangi ay olduğunu, `Now` şimdiki zamanı, `Today` bu günün tarihini, `Year` o anın hangi yıl olduğunu tutar. `Now` ve `Today`, elbette bilgisayarın gösterdiği tarih ve zamanı tutar. Ayrıca Now o anın bütün bileşenlerini tuttuğu için, `DateTime` yapısında olan her öğeye sahiptir.

DateTime Yapısının Metotları

`DateTime` yapısında, tarih ve zaman ile ilgili işlemleri yapan 50 ye yakın metot vardır. Bazılarının işlevlerini söylemekle yetineceğiz. Tarih ve zaman ile ilgili program yazanlar, `msdn` kütüphanesinden bu metotları alabilirler.

`AddDays` metodu bir tarihe belirli sayıda gün eklenince (veya çıkarılınca) yeni tarihin ne olacağını bulur. `Compare` metodu iki tarihi mukayese eder, hangisinin önce olduğunu bulur. `GetDateTimeFormats` metodu tarihi istenen biçemdeki metne dönüştürür. `IsLeapYear` metodu o yılın artık yıl olup olmadığını bulur. `Parse` metodu tarih ve zaman belirten bir stringi `DateTime` eşdeğerine dönüştürür. `ToShortDateString` metodu tarihi kısa biçemli stringe dönüştürür (Parse metodunun tersidir). `ToLongDateString` metodu tarihi uzun biçemli stringe dönüştürür (Parse metodunun tersidir).

DateTime Yapısının Operatörleri

Tarih ve zaman ile ilgili işlemleri yapmaya yarayan sekiz operatör şunlardır: `Addition`, `Equality`, `GreaterThan`, `GreaterThanOrEqual`, `Inequality`, `LessThan`, `LessThanOrEqual`, `Subtraction`. Bunların ne yaptıkları adlarından belli olduğu için, açıklamalarına girmeyecek, ancak gerekli olduklarında kullanacağız.

DateTime Yapısıyla İlgili Arayüzler (Interfaces)

Tarih ve zaman ile ilgili çok sayıda arayüz vardır. Onları burada açıklamaya gerek görmüyoruz.

ToString() Metodu ile Tarih Yazdırma

Hemen her sınıfta var olan `ToString()` metodu `DateTime` yapısının öğelerini standart çıkışa belirtilen biçemde yazar. Bu metodun 'overloaded' olmuş dört farklı şeklini kullanacağız:

```
public string ToString();

public string ToString(IFormatProvider);

public string ToString(string);

public string ToString(string, IFormatProvider);
```

Parametresiz olan birincisi tarihi biçemlemeden yazar. İkincisi `IformatProvider` arayüzünü (interface) parametre olarak alır. Bu arayüz `NumberFormatInfo`, `DateTimeFormatInfo` ve `CultureInfo` sınıfları tarafından çıktıyı biçemlemek için kullanılır. Üçüncüsü string tipi parametre alır. Dördüncüsü bir string tipi ve bir `IformatProvider` olmak üzere iki parametre alır. Belirtgen adını alan `string` tipi parametreler aşağıdaki tabloda gösterilmiştir.

Belirtke	**DateTimeFormatInfo özelliği**	**Pattern değeri (TR için)**
t	ShortTimePattern	h:mm tt
d	ShortDatePattern	M/d/yyyy
T	LongTimePattern	h:mm:ss tt
D	LongDatePattern	dddd, MMMM dd, yyyy
f	(combination of D and t)	dddd, MMMM dd, yyyy h:mm tt
F	FullDateTimePattern	dddd, MMMM dd, yyyy h:mm:ss tt
g	(combination of d and t)	M/d/yyyy h:mm tt
G	(combination of d and T)	M/d/yyyy h:mm:ss tt
m, M	MonthDayPattern	MMMM dd
y, Y	YearMonthPattern	MMMM, yyyy
r, R	RFC1123Pattern	ddd, dd MMM yyyy HH':'mm':'ss 'GMT' (*)
s	SortableDateTimePattern	yyyy'-'MM'-'dd'T'HH':'mm':'ss (*)
u	UniversalSortableDateTimePattern	yyyy'-'MM'-'dd HH':'mm':'ss'Z' (*)
		(*) = ülkeden bağımsız

`Today` özgeninin tuttuğu tarih, `ToString()` metodu tarafından konsola `"gün-ay-yıl"` biçeminde, `"ay-gün-yıl"` biçeminde veya istenen başka bir biçemde yazdırılabilir.

Tarih01.cs

```
using System;

namespace Methods
{
    class Tarih01
    {
```

```
        static void Main()
        {
            Console.WriteLine(DateTime.Today.ToString("dd MMMM yy"));
        }
    }
}
```

Çıktı

19 Temmuz 08

Uyarı

```
DateTime.Today.ToString("dd MMMM yy")
```

deyimi yerine

```
DateTime.Now.ToString("dd MMMM yy")
```

deyimini yazarsanız, aynı çıktıyı elde ederiniz. Çünkü `Today` ve `Now` özgenlerinin her ikisi de bugün'ün tarih bileşenine sahiptir. Ama `Now` özgeni daha çok bileşen tutar. Gerçekte o şimdiki zamanı `DateTime`'in belirlediği bütün bileşenlerine ayırabilir. Örneğin, saat, dakika, saniye, milisaniye bileşenleri Now tarafından tutulur, ama Today tarafından tutulmaz.

Bu günün tarihi yerine başka bir tarih kullanabiliriz. Önce `DateTime` sınıfından işaretçisi `dt` olan bir nesne yaratacak ve ona 29-10-1923 değerini vereceğiz. Bu işi yapan deyim şudur:

```
DateTime dt = new DateTime(1923, 10, 29 );
```

dt nin işaret ettiği nesneye (1923,10,29) şeklinde atadığımız tarih o nesne içinde DateTime tipi tarihe dönüşmüştür. Onun üzerinde artık tarih ile ilgili her işlemi yapabilir ve istediğimiz string biçemine dönüştürebiliriz. Örneğin, bu tarihi ToString metodu ile "29 Ekim 1923" biçemine çevirelim.

```
dt.ToString(”dd MMMM yy “)
```

Bu söylediklerimizi bir araya getirirsek, aşağıdaki programı elde ederiz.

Tarih02.cs

```
using System;

namespace Methods
{
    class Tarih02
    {
        static void Main()
        {
            DateTime dt = new DateTime(1923, 10, 29);
            Console.WriteLine(dt.ToString("dd MMMM yy "));
        }
    }
}
```

Çıktı

29 Ekim 1923

Yöresellik (locale)

`ToString()` metodu tarih yazarken string tipi parametre olarak yazılan "dd MMMM yy" biçemine uydu. Ay adını `Türkçe` yazdı: `Temmuz`. Peki ama, Türkçe bilmeyen birisi için bu çıktı uygun mu?

.NET bütün ülke ve kültürlerin kullandıkları farklı biçemlerde çıktı verme yeteneğine sahiptir. Kullandığımız Windows işletim sistemi Türkçe'ye ayarlı olduğu için, .NET tarih ve sayı çıktılarını bizim kullandığımız biçeme dönüştürerek veriyor. Windows başka bir dile ayarlı ise, çıktılar o dilin *syntax*'ına uyar. Bu oldukça iyi bir yetenektir. Ama, işletim sistemimiz hangi dilde olursa olsun, çıktının belirli bir yöresel biçemde yazılmasını isteyebiliriz. .NET bunu da yapar.

Bu söylediğimizi yaptırmak için, uygulamamızın çalıştığı `thread'` in `CurrentCulture` özgenini istediğimiz ülke ve kültüre ayarlamamız yetecektir. Örneğin, yukarıdaki tarih çıktısını Almanca yazdırmak için

```
Thread.CurrentThread.CurrentCulture = new CultureInfo("de-DE");
```

deyimini programımıza eklememiz yetecektir.

Tarih03.cs

```
using System;
using System.Globalization;
using System.Threading;

namespace Methods
{
    class Tarih03
    {
        static void Main()
        {
            Thread.CurrentThread.CurrentCulture = new CultureInfo("de-
DE");
            Console.WriteLine(DateTime.Today.ToString("MMMM dd yyyy"));
        }
    }
}
```

Çıktı

```
Juli 19 2008
```

Tarihi İspanya'nın Bask bölgesinde kullanılan biçemde yazdırmak isterseniz, programda aşağıdaki değişikliği yapınız.

```
CultureInfo("eu-ES")
```

çıktının

```
uztaila 19 2008
```

olduğunu göreceksiniz.

Aşağıdaki program, aynı işi program koşarken kültür thred'ini değiştirerek tekrarlamaktadır.

Tarih04.cs

```
using System;
using System.Globalization;
using System.Threading;

namespace Yöresellik
{
    class Tarih04
    {
        static void Main(string[] args)
        {
            DateTime d = new DateTime(1923, 10, 29);

            // etkin kültüre (türkçe)
```

```
            Console.WriteLine(d.ToLongDateString());

            // IformatProvider kullanarak Avusturya Almancasına geçiş
            Console.WriteLine(d.ToString("D", new CultureInfo("de-AT")));

            // etkin kültür thread'ini (Türkçe) kullan
            CultureInfo cInf = Thread.CurrentThread.CurrentCulture;
            Console.WriteLine(cInf.ToString() + ": " + d.ToString("D"));

            // IformatProvider kullanarak İspanyol kültür thread'ine
geçiş
            cInf = new CultureInfo("es-ES");
            Thread.CurrentThread.CurrentCulture = cInf;
            Console.WriteLine(cInf.ToString() + ": " + d.ToString("D"));
        }
    }
}
```

Çıktı

```
29 Ekim 1923 Pazartesi
Montag, 29. Oktober 1923
tr-TR: 29 Ekim 1923 Pazartesi
es-ES: lunes, 29 de octubre de 1923
```

Bu çıktıyı açıklayalım.

```
DateTime d = new DateTime(1923, 10, 29);
```

deyimi `DateTime` sınıfının `d` ile işaret (referans) edilen bir nesnesini yaratmıştır. Bu nesne "`29-10-1923`" tarihini tutan `DateTime` tipindendir. (Class'ın soyut bir veri tipi olduğunu anımsayınız.)

İlk satırı yazan

```
Console.WriteLine(d.ToLongDateString());
```

deyimidir. `ToLongDateString()` metodu, `d` nin işaret ettiği nesnenin tuttuğu tarihi, çıktının ilk satırında olduğu gibi yazdırır.

İkinci satır çıktısında, kültür thread'i Avusturya Almancasına geçmiştir. Bunu yapan

```
d.ToString("D", new CultureInfo("de-AT"))
```

deyimidir. ToString() metodunun ikinci parametresi olan

```
new CultureInfo("de-AT")
```

deyimi, `CultureInfo` sınıfının bir nesnesini yaratmakta ve onu Avusturya Almancası thredine ayarlamaktadır. Bu thread, tarihi o yörenin kullandığı biçimde bir string'e dönüştürür. Artık, `WriteLine()` metodu o stringi konsola göndermekle yükümlüdür.

Üçüncü satır çıktısından önce

```
CultureInfo cInf = Thread.CurrentThread.CurrentCulture;
```

deyimi, threadi etkin olana, yani Türkçe threade tekrar döndürmektedir. Dolayısıyla üçüncü satır Türkçe biçemindedir.

Dördüncü satırda ise İspanyolca thread'e geçilmektedir.

Yukarıdaki programda sabit bir tarih kullandık. İstersek, sabit bir tarih yerine bu günün tarihini koyabiliriz. Bunun için `Main()` metodunun gövdesindeki ilk satır yerine aşağıdaki iki deyimi koymak yetecektir.

```
DateTime d = new DateTime();
d = DateTime.Today;
```

Bunu yapınca, çıktı şuna benzeyecektir. Tabii, çıktıdaki tarih, programın koşturulduğu tarih olacaktır.

Çıktı

```
21 Temmuz 2008 Pazartesi
Montag, 21. Juli 2008
tr-TR: 21 Temmuz 2008 Pazartesi
es-ES: lunes, 21 de julio de 2008
```

Aşağıdaki program, Türkçe tarih biçemleri yazdırma yöntemlerini göstermektedir.

Tarih06.cs

```
using System;

namespace BiçemliÇıktılar
{
    class Tarih06
    {
        static void Main()
        {
            Console.WriteLine(DateTime.Now);
            Console.WriteLine(DateTime.Now.ToString());
            Console.WriteLine(DateTime.Now.ToShortTimeString());
            //Console.WriteLine(DateTime.Now.ToDateString());
            Console.WriteLine(DateTime.Now.ToLongTimeString());
            Console.WriteLine(DateTime.Now.ToLongDateString());
            Console.WriteLine(DateTime.Now.ToString("d"));
            Console.WriteLine(DateTime.Now.ToString("D"));
            Console.WriteLine(DateTime.Now.ToString("f"));
            Console.WriteLine(DateTime.Now.ToString("F"));
            Console.WriteLine(DateTime.Now.ToString("g"));
            Console.WriteLine(DateTime.Now.ToString("D"));
            Console.WriteLine(DateTime.Now.ToString("m"));
            Console.WriteLine(DateTime.Now.ToString("r"));
            Console.WriteLine(DateTime.Now.ToString("s"));
            Console.WriteLine(DateTime.Now.ToString("t"));
            Console.WriteLine(DateTime.Now.ToString("T"));
            Console.WriteLine(DateTime.Now.ToString("u"));
            Console.WriteLine(DateTime.Now.ToString("U"));
            Console.WriteLine(DateTime.Now.ToString("y"));
            Console.WriteLine(DateTime.Now.ToString("dddd, MMMM dd
yyyy"));
            //Console.WriteLine(DateTime.Now.ToString("ddd, MMM d" '
"yy"));
            Console.WriteLine(DateTime.Now.ToString("dddd, MMM dd"));
            Console.WriteLine(DateTime.Now.ToString("M/yy"));
            Console.WriteLine(DateTime.Now.ToString("dd-MM-yy"));
        }
    }
}
```

Çıktı

```
23.07.2008 00:37:25
23.07.2008 00:37:25
00:37
```

```
00:37:25
23 Temmuz 2008 Çarşamba
23.07.2008
23 Temmuz 2008 Çarşamba
23 Temmuz 2008 Çarşamba 00:37
23 Temmuz 2008 Çarşamba 00:37:25
23.07.2008 00:37
23 Temmuz 2008 Çarşamba
23 Temmuz
Wed, 23 Jul 2008 00:37:25 GMT
2008-07-23T00:37:25
00:37
00:37:25
2008-07-23 00:37:25Z
22 Temmuz 2008 Salı 21:37:25
Temmuz 2008
Çarşamba, Temmuz 23 2008
Çarşamba, Tem 23
7.08
23-07-08
```

String.Format() Metodu ile Tarih Biçemleme

Buraya kadar ToString() metodu ile tarihi string olarak biçemlemeyi öğrendik. Bu kesimde String sınıfının Format() metodunu kullanarak benzer işleri yapacağız.

Tarih07.cs

```
using System;

namespace TarihVeZaman
{
    class Tarih07
    {
        static void Main()
        {
            DateTime dt = new DateTime(2008, 7, 13, 23, 35, 03, 456);

            Console.WriteLine(String.Format("{0:y yy yyy yyyy}", dt));
            Console.WriteLine(String.Format("{0:M MM MMM MMMM}", dt));
            Console.WriteLine(String.Format("{0:d dd ddd dddd}", dt));
            Console.WriteLine(String.Format("{0:h hh H HH}", dt));
            Console.WriteLine(String.Format("{0:m mm}", dt));
            Console.WriteLine(String.Format("{0:s ss}", dt));
            Console.WriteLine(String.Format("{0:f ff fff ffff}", dt));
            Console.WriteLine(String.Format("{0:F FF FFF FFFF}", dt));
            Console.WriteLine(String.Format("{0:t tt}", dt));
            Console.WriteLine(String.Format("{0:z zz zzz}", dt));
        }
    }
}
```

Bu programın çıktısı ve açıklamaları aşağıdaki tabloda verilmiştir. Tabloyu programla satır satır karşılaştırarak istenen biçemin nasıl elde edildiğini görünüz.

Biçemleme	**Çıktı**	**Adı**
String.Format("{0:y yy yyy yyyy}", dt);	8 08 008 2008	yıl
String.Format("{0:M MM MMM MMMM}", dt);	7 07 Tem Temmuz	ay
String.Format("{0:d dd ddd dddd}", dt);	13 13 Paz Pazar	gün
String.Format("{0:h hh H HH}", dt);	11 11 23 23	Saat 12/24
String.Format("{0:m mm}", dt);	35 35	dakika
String.Format("{0:s ss}", dt);	3 03	saniye
String.Format("{0:f ff fff ffff}", dt);	4 45 456 4560	sn/1000
String.Format("{0:F FF FFF FFFF}", dt);	4 45 456 456	Sn/1000, sıfırsız
String.Format("{0:t tt}", dt);		A.M. veya P.M.
String.Format("{0:z zz zzz}", dt);	+3 +03 +03:00	Zaman dilimi

`String.Format()` metodu ile tarih biçemlerken, aşağıda belirtilen standart biçem belirtgenlerini kullanırız.

Tarih08.cs

```
using System;

namespace TarihVeZaman
{
    class TarihYazma02
    {
        static void Main()
        {
            DateTime dt = new DateTime(2008, 7, 13, 23, 35, 03, 456);

            Console.WriteLine(String.Format("{0:t}", dt));
            Console.WriteLine(String.Format("{0:d}", dt));
            Console.WriteLine(String.Format("{0:T}", dt));
            Console.WriteLine(String.Format("{0:D}", dt));
            Console.WriteLine(String.Format("{0:f}", dt));
            Console.WriteLine(String.Format("{0:F}", dt));
            Console.WriteLine(String.Format("{0:g}", dt));
            Console.WriteLine(String.Format("{0:G}", dt));
            Console.WriteLine(String.Format("{0:m}", dt));
            Console.WriteLine(String.Format("{0:y}", dt));
```

```
            Console.WriteLine(String.Format("{0:r}", dt));
            Console.WriteLine(String.Format("{0:s}", dt));
            Console.WriteLine(String.Format("{0:u}", dt));
        }
    }
}
```

Çıktı

```
23:35
13.07.2008
23:35:03
13 Temmuz 2008 Pazar
13 Temmuz 2008 Pazar 23:35
13 Temmuz 2008 Pazar 23:35:03
13.07.2008 23:35
13.07.2008 23:35:03
13 Temmuz
Temmuz 2008
Sun, 13 Jul 2008 23:35:03 GMT
2008-07-13T23:35:03
2008-07-13 23:35:03Z
```

Standart Biçem Belirtgenleri

Kod	Çıktısı	.NET adı
String.Format("{0:t}", dt);	23:35	ShortTime
String.Format("{0:d}", dt);	13.07.2008	ShortDate
String.Format("{0:T}", dt);	23:35:03	LongTime
String.Format("{0:D}", dt);	13 Temmuz 2008 Pazar	LongDate
String.Format("{0:f}", dt);	13 Temmuz 2008 Pazar 23:35	LongDate+ShortTime
String.Format("{0:F}", dt);	13 Temmuz 2008 Pazar 23:35:03	FullDateTime
String.Format("{0:g}", dt);	13.07.2008 23:35	ShortDate+ShortTime
String.Format("{0:G}", dt);	13.07.2008 23:35:03	ShortDate+LongTime
String.Format("{0:m}", dt);	13 Temmuz	MonthDay
String.Format("{0:y}", dt);	Temmuz 2008	YearMonth

String.Format("{0:r}", dt);	Sun, 13 Jul 2008 23:35:03 GMT	RFC1123
String.Format("{0:s}", dt);	2008-07-13T23:35:03	SortableDateTime
String.Format("{0:u}", dt);	2008-07-13 23:35:03Z	UniversalSortableDateTime

Tarih09.cs

```
using System;

namespace TarihVeZaman
{
    class TarihYazma02
    {
        static void Main()
        {
            DateTime dt = new DateTime(2008, 7, 13, 23, 35, 03, 456);

            Console.WriteLine(String.Format("{0:M/d/yyyy}", dt));
            Console.WriteLine(String.Format("{0:MM/dd/yyyy}", dt));
            Console.WriteLine(String.Format("{0:ddd, MMM d, yyyy}", dt));
            Console.WriteLine(String.Format("{0:dddd, MMMM d, yyyy}",
dt));
            Console.WriteLine(String.Format("{0:MM/dd/yy}", dt));
            Console.WriteLine(String.Format("{0:MM/dd/yyyy}", dt));
            Console.WriteLine(String.Format("{0:d/M/yyyy HH:mm:ss}",
dt));
            Console.WriteLine(String.Format("{0:d/M/yyyy HH:mm:ss}",
dt));
        }
    }
}
```

Çıktı

```
7.13.2008
07.13.2008
Paz, Tem 13, 2008
Pazar, Temmuz 13, 2008
07.13.08
07.13.2008
13.7.2008 23:35:03
13.7.2008 23:35:03
```

sıfır öntakısız gün ve ay yazma

String.Format("{0:M/d/yyyy}", dt);	7.13.2008
String.Format("{0:MM/dd/yyyy}", dt);	07.13.2008

gün ve ay adlarını yazma

`String.Format("{0:ddd, MMM d, yyyy}", dt);`	`Paz, Tem 13, 2008`
`String.Format("{0:dddd, MMMM d, yyyy}", dt);`	`Pazar, Temmuz 13, 2008`

2 veya 4 haneli yıl yazma

`String.Format("{0:MM/dd/yy}", dt);`	`07.13.08`
`String.Format("{0:MM/dd/yyyy}", dt);`	`07.13.2008`

gün-ay ayracı ["/" dan "." ya geçer]

String.Format("{0:d/M/yyyy HH:mm:ss}", dt);	13.7.2008 23:35:03	- english (en-US)
String.Format("{0:d/M/yyyy HH:mm:ss}", dt);	13.7.2008 23:35:03	german (de-DE)

Aşağıdaki programı çözümleyiniz. Kodlarla çıktıları karşılaştırınız. Bazı kültürler için çıktıyı elde etmek için System.Globalization etkin kılınmalıdır. Aynı kodlardan farklı İşletim Sistemlerinde farklı çıktılar oluşur.

Tarih10.cs

```
using System;
//using System.Text;
//using System.Globalization;

namespace ex6
{
    class Program
    {
        static void Main(string[] args)
        {

            // Date biçemleri
            DateTime dt = DateTime.Now;
            // Standards
            // O or o  yyyy'-'MM'-'dd'T'HH':'mm':'ss'.'fffffffzz
            Console.WriteLine("{0:O}", dt);
            // R or r ddd, dd MMM yyyy HH':'mm':'ss 'GMT'
            Console.WriteLine("{0:R}", dt);
            // s yyyy'-'MM'-'dd'T'HH':'mm':'ss
            Console.WriteLine("{0:s}", dt);
            // u yyyy'-'MM'-'dd HH':'mm':'ss'Z'
            Console.WriteLine("{0:u}", dt);

            //  date/time belirteçleri
            // short time
            Console.WriteLine("{0:t}", dt);
            // long time
```

```
            Console.WriteLine("{0:T}", dt);
            // short date
            Console.WriteLine("{0:d}", dt);
            // long date
            Console.WriteLine("{0:D}", dt);
            // long date / short time
            Console.WriteLine("{0:f}", dt);
            // long date / long time
            Console.WriteLine("{0:F}", dt);
            // short date / short time
            Console.WriteLine("{0:g}", dt);
            // short date / long time
            Console.WriteLine("{0:G}", dt);
            // Round Trip
            Console.WriteLine("{0:o}", dt);

            // Kültüre göre değişenler
            Console.WriteLine("{0:dd/mm/yyyy HH:MM:ss}", dt);
            Console.WriteLine("{0:mm/dd/yyyy HH:MM:ss}", dt);
            Console.WriteLine("{0:yyyy/mm/dd HH:MM:ss}", dt);

            Console.WriteLine("{0:dd MMM yyyy HH:MM:ss}", dt);
            Console.WriteLine("{0:MMM dd yyyy HH:MM:ss}", dt);
            Console.WriteLine("{0:yyyy MMM dd HH:MM:ss}", dt);
            Console.ReadKey();
        }
    }
}
```

Çıktı

```
2008-08-29T18:19:46.3906250+03:00
Fri, 29 Aug 2008 18:19:46 GMT
2008-08-29T18:19:46
2008-08-29 18:19:46Z
18:19
18:19:46
29.08.2008
29 Ağustos 2008 Cuma
29 Ağustos 2008 Cuma 18:19
29 Ağustos 2008 Cuma 18:19:46
29.08.2008 18:19
29.08.2008 18:19:46
2008-08-29T18:19:46.3906250+03:00
29.19.2008 18:08:46
19.29.2008 18:08:46
2008.19.29 18:08:46
29 Ağu 2008 18:08:46
Ağu 29 2008 18:08:46
2008 Ağu 29 18:08:46
```

KAYNAKLAR

1. Andrew Troelsen: C# and the .NET Platform, Second Edition, Appress, 2007.
2. Borland C#BUILDER, Borland, 2003.
3. Bradley L. Jones: Sams Teach Yourself C# in 21 Days
4. Bruce Eckel: Thinking in C#, Prentice Hall, e-book.
5. Faraz Rasheed: Programmer's Heaven C# School, e-book.
6. http://flint.cs.yale.edu/cs112/lecture.html
7. http://www.academicresourcecenter.net/curriculum/pfv.aspx?ID=6001
8. http://www.dotnetngene.kr/NewTech/Temp/Lectures.aspx?
9. http://www.ssw.uni-linz.ac.at/Teaching/Lectures/CSharp/Tutorial/
10. J.Foxall – W.Haro-Chun: Sams Teach Yourself C# in 24 Hours.
11. Joel Murach: Murach's C# 2005, Murach.
12. Joseph Albahari - Ben Albahari: C# 3.0 in a Nutshell, Third Edition, O'Reilly, 2007.
13. msdn: Microsoft Developer Network, http://msdn.microsoft.com/en-us/default.aspx
14. Sefer Algan: Her Yönüyle C# (7.Basım), Pusula Yayıncılık, 2008.
15. V.Mukhi- S.Shanbhag- S.Mukhi: C# The Basics, e-book.